Revere-AMU System Architecture

ALPHA RELEASE

This document is an Alpha specification of the Revere-AMU System Architecture.
This document is for review purposes only and should not be used for any implementation,
as significant changes are likely. Furthermore, there are areas which are still
under investigation and incomplete at this time.

ARM-IHI-0078A ALP-03_b

Release information

Date	Version	Changes
2019/Feb/01	ALP-03_b	• Corrected the formatting of some tables (REVERE-1246) • Non confidential version
2019/Jan/17	ALP-03	• Registers and command messages hyperlinks • Add error detection, reporting and handling • Add interrupt for error reporting (REVERE-548) • Fixed LOG2_SLOT width (REVERE-600) • Moved TYPEB_TBL_BASE to the AMI-SW and aligned it to 4KB (REVERE-599) • Plan B for AMI discovery (REVERE-575) • Write up for unpinned memory access and shared virtual memory (REVERE-574) • Added capability to obtain a list of mapped AMIs to a given Function (REVERE-575) • Added a min/max message length parameters (REVERE-336) (REVERE-501) • Added the overprovisioning scenario • MPIDR and stashing to other caches (REVERE-289) • Removed MAX_AMI_HW register and added PF-PROBE-AHA • Added the AHA chaining scenario • Added Power Management chapter • Added section on AMI-HW • Wake-on-LAN use case (REVERE-323) • Clarify the table pointed to by MGT_TYPEB_BASE (REVERE-861) • Specify management commands opcodes (REVERE-917) • Added exception message for trapped commands • Clarify reset behavior w.r.t AMSes • Updating SAVE commmands (REVERE-1130) • Address the case where a PF command is sent through a VF management AMI (REVERE-1146) • Fix TX_DIGEST bits computation (REVERE-1169) • Clarify digest behavior (REVERE-470 and 1145) • Moving ASN creation by PF AMI ID arguments to PHY • REVERE-1206: clarified perf recommandation for SMMU and cleaned up IOVA • Add DMA summary table (REVERE-1197) • REVERE-1228: bitfield alignment between PF/F-AMS-RING-CONFIGURE • Expand overprovisioning use cases (REVERE-1211) • AMS not allowed to be connected with more than one ASN (REVERE-1235) • PF-ASN-Destroy response now uses PHY_ID of AMI (REVERE-1234) • Enabling AMI/AMS saving by PF without unmapping first, refactored state machine, fixed typos and scenario (REVERE-1193) • Fixed Type A register access attributes (REVERE-1168 and REVERE-1123) • Refresh interrupts (REVERE-1145 and REVERE-1015) • Allow more interrupt vectors (REVERE-1229) • Added information about indices and PBA bits handling in the Live migration scenario (REVERE-1239 and REVERE-1241)

Date	Version	Changes
2018/Apr/12	ALP-02	• Moved from table-based to message-based configuration (REVERE-527) • Use dedicated rings for PF/VF management rather than AMI-SW (REVERE-503) • Removed partitions, AGC and AVC as concepts. Made PF and VFs symmetric (REVERE-505) • Clarify PCI MSI-X per vector mask (REVERE-544) • Moved commands/responses to separate chapter and restructured document (REVERE-460) • Modified some of the architectural parameter sizes (AMI_PER_VF, MAX_AMI_SW, MAX_AMI_HW and MAX_ASN) (REVERE-491) • Renamed substream ID to PASID and removed Stream ID • Added the description of the profiling and tracing capabilities • Detail BAR space map (REVERE-557) • PCIe spec should be PCI Express Base Specification, Rev. 4.0 Version 0.9 (REVERE-534) • Change reference to Section 7.8 in the PCI Express Base Specification (REVERE-537) • Use Device ID and Vendor ID, et al. (REVERE-537) • Revere is based on PCI Express, and not PCI (REVERE-532) • Align introduction with latest presentation (REVERE-560) • Allow to drop VF response to prevent DoS (REVERE-437) • Make PF and VF management interfaces a special case (REVERE-503) • Clarify Per-Vector Masking (REVERE-544) • AMS Data Structures updated (REVERE-260) • Added Over-writing AMS (REVERE-542) • MFO Evolution - support for fixed size messages / embedded out-of-band buffer table (REVERE-558) • Added a exhaustive map of the BAR0 spaces • Live Migration use case (REVERE-496)

Date	Version	Changes
2017/Dec/19	ALP-01	• Add DIGEST_MASK registers (REVERE-345) • Add Tx and Rx vectors to the AMI SW Table (REVERE-149) • Add Interrupts write-up (REVERE-157) • Add Pin-level interface write-up (REVERE-157) • Add IRQ_CTRL registers to the AMI-SW • Rework thresholds for DIGEST bits and move them from the AMI_SW Table to the AMS • Programming sequence clarification (REVERE-327) • STOPPED clarification (REVERE-284) • Clarify use of VF Reset (REVERE-282) • Changed AMS entry format in the AMI table (REVERE-258) • Ring buffer clarifications (REVERE-378) • Alignment for ring buffer and suggest to encode buffer size as an integer (REVERE-308) • Exact format of the AMS data structure (REVERE-260) • Support larger ring sizes (REVERE-350) • Clarified how AMS relates to (or lives within) an AMI (REVERE-292) • Changed VF_AMI_SW to TYPEB_AMI_SW (REVERE-330) • Added F_OWNER in AMI-HW, renamed VF_ID to F_OWNER in AMI-SW table • AGC, AVC: all registers are 64-bit (REVERE-339) • Rearranged AMI-SW/AMI-HW tables, aligned the beginning of the AMS entries to 64B • Message format is now a section (REVERE-286) • Added a out-of-band buffer length to MFO3 (REVERE-288) • Clarified the range of LENGTH for the in-band payload size (REVERE-291) • Added ordering requirements for the SW interface (REVERE-280) • Added the description of a basic example
2017/Nov/07	ALP-00e	• Redrew introduction figures and cleaned up terminology (REVERE-331) • AMI Table invalidate clarification (REVERE-324) • Clarified various statements regarding AGC and AMI-SW (REVERE-325 and REVERE-326) • Added DIGEST threshold (REVERE-298) • Corrected misc bugs/typos (REVERE-329) • Clarified table rules and programming sequences (REVERE-327) • Clarified behavior of an AMS that is disabled (REVERE-284) • Defined completion semantics for table invalidates (REVERE-283) • Clarified reset behavior (REVERE-282) • AMI-SW0 location in PF memory space (REVERE-318)

Date	Version	Changes
2017/Aug/07	ALP-00	• First ALPHA release. • Removed OFFSET in the AMI_SW table entries (updated corresponding diagrams) • Renamed AMI_ID to LOCAL_AMI_ID in the AMI_SW Table entries • Removed the VF_AMI_TABLE • Added PASID/PASID_ENABLE and TYPE in the AMI_SW Table • Added VF_AMI_SW_TABLE for type B AMS • Modified the AVC partition • Updated the base pointer and size to reflect changes (removal of VF_AMI_TABLE and addition of VF_AMI_SW_TABLE) • Added AMI-HW table • Modified the AGC partition • Removed MAX_AMI register • Added MAX_AMI_SW register • Added MAX_AMI_HW register • Removed NUM_TYPE_H register • Removed AMI_TABLE_BASE and AMI_TABLE_SIZE • Added AMI_SW_TABLE_BASE and AMI_SW_TABLE_SIZE • Added AMI_HW_TABLE_BASE and AMI_HW_TABLE_SIZE
2017/Jun/28	DEV-01	• First draft. • Document structure • Message formats and cache stashing sections added.

Non-Confidential Proprietary Notice

Contents

Revere-AMU System Architecture

Preface

About this book

The Revere-AMU System Architecture defines a new system component for device assignment, the Accelerator Management Unit (AMU). The AMU supports message passing between software and hardware tasks using architected interfaces described in this document: a memory-mapped device for software and a pin-level interface for hardware.

This manual has the following parts:

Part A

Specification of Revere-AMU System Architecture.

Part B

Pin-level interface for hardware agents.

Part C

Use cases.

Using this book

The information in this manual is organised into parts, as described in this section.

Part A: Revere-AMU System Architecture

Part A describes the architecture of `Revere-AMU System`. It contains the following chapters:

Chapter A1 *Scope*

Provides a description of the scope of the Revere-AMU System architecture.

Chapter A2 *Introduction*

Provides an introduction of the Revere-AMU System Architecture.

Chapter A3 *Architecture*

Specifies Revere-AMU architecture.

Chapter A4 *Management Commands and Responses*

Describes the format and behaviour of management commands and responses.

Chapter A5 *Exceptions*

Describes the format and behaviour of exceptions generated by the AMU.

Part B: Pin-level interface for hardware agents

Part B describes the optional pin-level interface for hardware agents.

Chapter B1 *Scope*

Provides a description of the scope of the optional pin-level interface for hardware agents.

Chapter B2 *AMU AHA Interface protocol*

Specifies the AMU AHA Interface (AAI) protocol and describes the components of an AAI-compliant implementation.

Chapter B3 *AHA DMA Transport Layer*

Provides a description of the optional DMA interface to access out-of-band data.

Chapter B4 *Power control*

Describes pin-level interface power control.

Part C: Use cases

Part C describes some use cases of `Revere-AMU System`. It contains the following chapters:

Chapter C1 *Null accelerator*.

Provides a description of a basic use case for the use of Revere-AMU.

Chapter C2 *VM Live Migration*

Describes a use case where a Virtual Function is migrated between two physical machines.

Chapter C3 *Overprovisioning*

Describes a use case where more AMIs are presented to software than what is effectively implemented.

Chapter C4 *AHA chaining*

Describes a use case where two accelerator contexts are chained.

Chapter C5 *Lightweight*

Provides a description of an use case leveraging the Revere-AMU capabilities targeted at a lightweight implementation.

Chapter C6 *Virtualization offload*

Provides a description of an use case where out-of-band data is transported across Virtual Machines.

Chapter C7 *Networking SoC*

Provides a description of a complex use case involving several AHA targeted at networking-specific applications.

Chapter C8 *Wake-on-LAN*

Provides a description of a complex use case involving a network interface AHA, which is used to wake up the system from low power mode in response to an incoming network packet.

Conventions

Typographical conventions

The typographical conventions are:

italic

Introduces special terminology, and denotes citations.

bold

Denotes signal names, and is used for terms in descriptive lists, where appropriate.

`monospace`

Used for assembler syntax descriptions, pseudocode, and source code examples.

Also used in the main text for instruction mnemonics and for references to other items appearing in assembler syntax descriptions, pseudocode, and source code examples.

SMALL CAPITALS

Used for some common terms such as IMPLEMENTATION DEFINED.

Used for a few terms that have specific technical meanings, and are included in the Glossary.

Red text

Indicates an open issue.

Blue text

Indicates a link. This can be

- A cross-reference to another location within the document
- A URL, for example http://infocenter.arm.com

Numbers

Numbers are normally written in decimal. Binary numbers are preceded by `0b`, and hexadecimal numbers by `0x`. In both cases, the prefix and the associated value are written in a monospace font, for example `0xFFFF0000`. To improve readability, long numbers can be written with an underscore separator between every four characters, for example `0xFFFF_0000_0000_0000`. Ignore any underscores when interpreting the value of a number.

Pseudocode descriptions

This book uses a form of pseudocode to provide precise descriptions of the specified functionality. This pseudocode is written in a monospace font. The pseudocode language is described in the Arm Architecture Reference Manual.

Assembler syntax descriptions

This book contains numerous syntax descriptions for assembler instructions and for components of assembler instructions. These are shown in a `monospace` font.

Additional reading

This section lists publications by Arm and by third parties.

See Arm Infocenter (http://infocenter.arm.com) for access to Arm documentation.

[1] *ARM Architecture Reference Manual for ARMv8-A architecture profile*. (ARM DDI 0487 A.i) ARM Ltd.

[2] *PCI Express Base Specification*. (Rev. 4.0 Version 1.0) PCI-SIG.

[3] *Arm Server Base System Architecture*. (ARM DEN 0029A 3.1) ARM Ltd.

[4] *AMBA® AXI and ACE Protocol Specification*. (ARM IHI 0022F) ARM Ltd.

[5] *ARM® AMBA® 5 CHI Architecture Specification*. (ARM IHI 0050B) ARM Ltd.

[6] *Arm Arm System Memory Management Unit Architecture Specification*. (ARM IHI 0070A 3.0, 3.1 and 3.2) ARM Ltd.

[7] *MPAM Extension Architecture v1.0*. (ECM-0485919 11.0-EAC4.0) ARM Ltd.

[8] *AMBA 4 AXI4-Stream Protocol Specification*. (ARM IHI 0051) ARM Ltd.

[9] *SMC CALLING CONVENTION, System Software on ARM® Platforms*. (ARM DEN 0028B) ARM Ltd.

[10] *ARM® Power State Coordination Interface, Platform Design Document*. (ARM DEN 0022D) ARM Ltd.

Feedback

Arm welcomes feedback on its documentation.

Feedback on this book

If you have comments on the content of this book, send an e-mail to errata@arm.com. Give:

- The title (Revere-AMU System Architecture).
- The number (ARM-IHI-0078A ALP-03_b).
- The page numbers to which your comments apply.
- The rule identifiers to which your comments apply, if applicable.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests PDFs only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the appearance or behavior of any document when viewed with any other PDF reader.

Part A
Architecture Specification

Chapter A1
Scope

This section is informative.

A1.1 Assigned and mediated devices

Computing systems increasingly include devices (for example: accelerators, I/O) that are *assigned* to the software that is using them. Device assignment means that the software using the device directly reads/writes memory mapped registers on the device, and/or directly reads/writes data structures that are accessed by the DMA facilities of the device.

The alternative to device assignment is *mediation* by a single device driver that has exclusive access to the physical device resources. Conventionally, the mediating device driver runs at a higher privilege level to other software, and implements a software abstraction of the device based on system calls. In the case of virtualization, the software abstraction is either based on hypercalls (known as para-virtualization) or trapping and emulating device accesses for other software using the device.

A1.2 Motivation for assignment of devices

Device assignment is motivated by a combination of system level requirements that are difficult or impossible to meet using mediating software. One such requirement is low latency access to the accelerator, where the cost of a system call is high relative to the granule of work offloaded to the accelerator. Another such requirement is isolation, where failure or compromise of the mediating software creates safety and/or security concerns.

Examples of devices where assignment is motivated include:

- Packet processing accelerators in a networking system, where the packet lifetime is short (1000s of cycles) and might involve several look-aside accelerators and packet I/O.
- Accelerators and CPUs that form a computer vision pipeline. The accelerator must have multiple directly assigned interfaces to meet safety requirements.

Generic device assignment is where software can manage device assignment without specific knowledge of that device. Generic assignment requires sufficient system architecture to define assignment controls such as address translation scheme and resets.

A small number of devices provide services to the operating system (for example: networking and storage I/O). With generic assignment, the majority of devices that do not provide an operating system service, do not require an operating system device driver.

A1.3 Supporting device assignment on Arm systems

Device assignment is supported in Arm systems using existing capabilities of the Arm CPU architecture and associated system architecture. For example, the virtual memory architecture for a PE [1] allows privileged software to enforce exclusive access of assigned software to a subset of device memory mapped registers at a page granule. The SMMU architecture allows device DMA to be translated to the virtual address space of the assigned software.

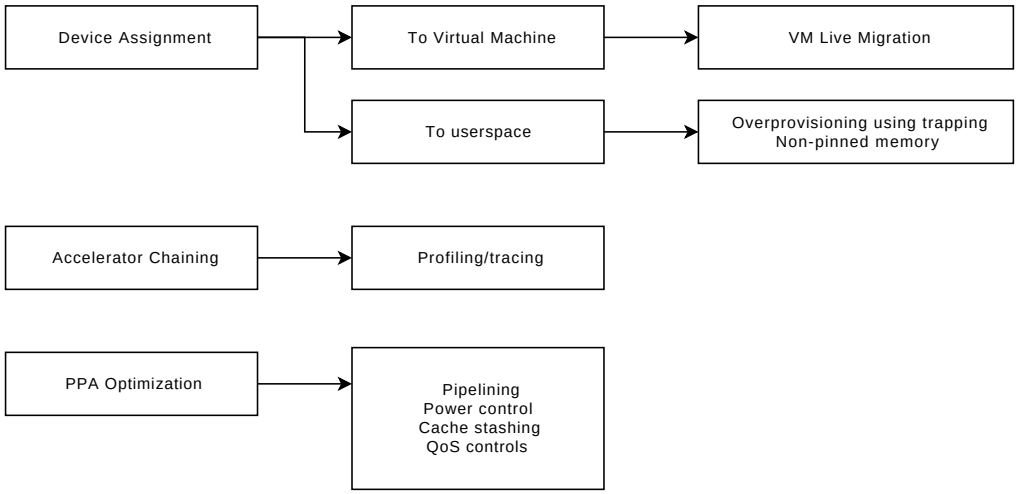

Figure A1.1: Requirements and derived requirements for a hardware/software interface, that can be met using the Revere-AMU system architecture

However, device assignment is complex and the requirements list is growing. Figure A1.1 summarises requirements for a HW/SW interface, including derived requirements, that can be met using tools provided in the Revere-AMU architecture.

Virtualization is increasingly used in systems, in which case device assignment to a virtual machine is required for performance and to avoid the need for device specific code in the hypervisor. To make use of the device as low-friction as possible, it's beneficial to support live migration of virtual machines. This requires the hypervisor to quiesce the device, and save/restore it's state.

Userspace device driver frameworks are also commonly used for performance and to enable the simpler development and deployment story of userspace drivers. Overprovisioning is beneficial to support many userspace processes, where an operating system driver dynamically swaps a finite number of accelerator contexts using a trapping scheme. Supporting non-pinned memory removes the burden on the userspace software to register needed buffers with the operating system.

Accelerator chaining supports direct interaction between accelerators without intermediation of software. Software must have controls to 'orchestrate' the chains of accelerators. It is highly beneficial for debug and performance tuning to have visibility of accelerator to accelerator communication, which requires profiling and tracing capabilities.

The need for power, performance and area optimization generates other requirements, for example:

- Pipelining: the ability to have multiple outstanding requests to/from the device in order to hide latency. This requires a data structure with multiple entries (such as a ring buffer) to be shared between software and the device.
- Data movement optimizations, such as cache stashing.
- QoS control in cases of multiple assignment, or virtualization.
- Power control.

A1.4 Revere and device assignment

Revere-AMU is a system architecture to support device assignment. The Revere-AMU architecture modularises the device data-path (that implements the special functionality of the device) from the hardware/software interface (the 'plumbing' that communicates with software).

This architecture defines a new AMU system component (Accelerator Management Unit). An AMU implementation provides a generic HW/SW interface with the above (and other) direct assignment features. The Revere-AMU architecture uses ordered and credited message channels as the abstraction between the generic HW/SW interface provided by the AMU and accelerator or I/O devices.

The AMU transports messages between contexts in the system. Contexts can be either software-based (a thread running on a CPU) or hardware-based (such as a stream of work on an IO device or an accelerator).

The Revere-AMU architecture provides standardization that reduces costs and time-to-market by:

- advancing the feature set for generic device assignment (as opposed to device specific, therefore reducing the instances where a specific operating system device driver is needed).
- increasing the amount of software that can be reused.

An AMU implementation allows reuse of hardware components, where components can leverage the AMU in providing a HW/SW interface that supports direct assignment, rather than designing the necessary complexity for each accelerator or I/O device.

The Revere-AMU architecture is intended to enable accelerated and differentiated systems: Arm will not require compliance to the Revere-AMU architecture. However, compliance will bring benefits such as an ecosystem that supports reuse of common HW/SW components. Revere enables implementations to optimize movement of control and data messages under that standard interface, improving performance and efficiency.

Revere supports isolation using virtual memory with minimal software complexity and overhead.

Chapter A2
Introduction

This section is informative.

A2.1 Overview

The Revere-AMU system architecture defines an AMU system component, which supports exchanging messages between software threads and hardware agents in the system (for example, an accelerator). Hardware agents in the system support multiple contexts, have a shared, coherent view of memory and support virtual memory (for example, by using an MMU or SMMU).

A device is logically separated into an AMU and multiple hardware agents (AHAs). The AMU provides the hardware/software interface and AHAs implement device specific functionality, for example accelerator datapaths, work distribution/scheduling and IO.

The messaging provided by the AMU supports a standard device assignment model where multiple independent software components (VMs, processes, threads) have direct (or passthrough) access to a dedicated context on that device for reasons of performance and isolation.

This section describes the device assignment model supported by Revere-AMU and how the Revere-AMU features are used to implement that model. The device assignment model supports simultaneous assignment of multiple device contexts to multiple software agents, including virtualization using a hypervisor.

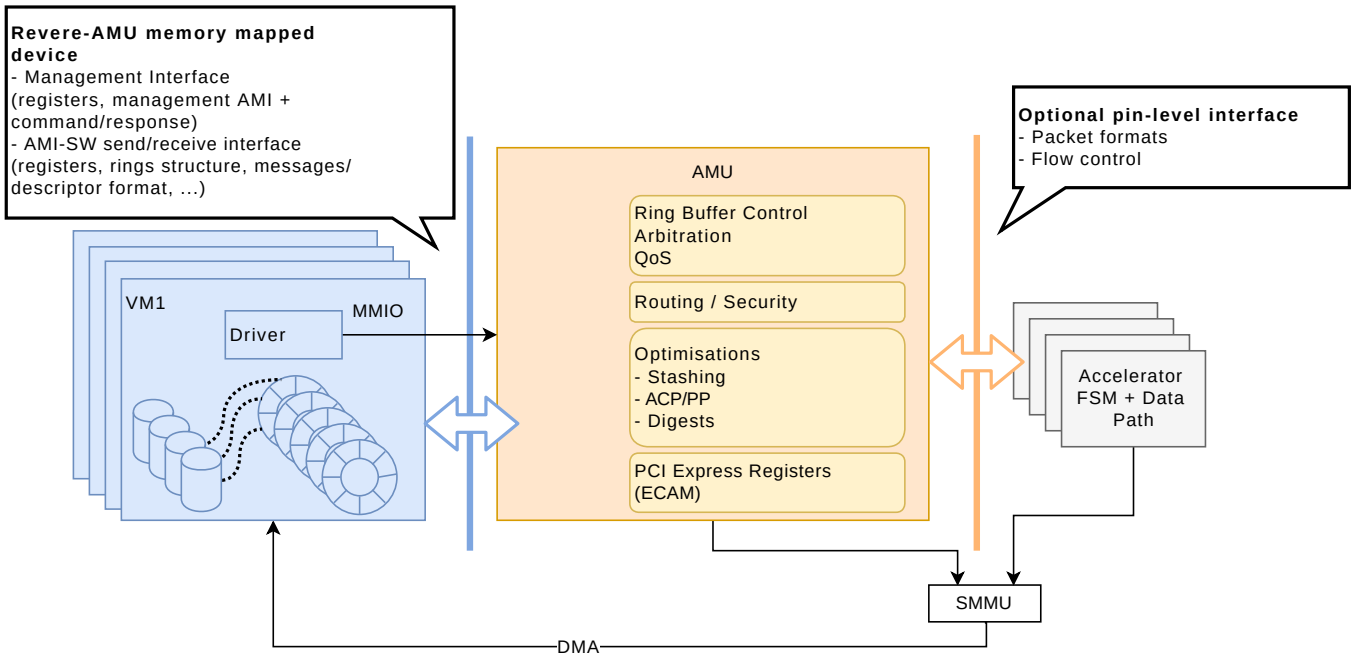

Figure A2.1: Scope of Revere System Architecture and AMU implementation. This architecture document describes a memory mapped device architecture and an optional Accelerator Message Interface for HW.

A2.1.1 Revere Messages

A message combines a descriptor with associated data. Except for a small set of architected message types, the semantics of a message is IMPLEMENTATION DEFINED. The purpose of architecting message passing is to enable hardware implementations of the AMU that optimize for performance and efficiency. Messages contain in-line data but can also contain references to data in the virtual address space relating to a context.

Note

The intention is that bulk data transfer still occurs using coherent shared memory and not as the payload of Revere-AMU messages.

A2.1.2 Accelerator Message Interface

Messages are sent and received using an architected Accelerator Message interface (AMI). This architecture defines two separate AMIs:

1. An Accelerator Message Interface (AMI-SW) to enable software running on an Armv8 PE to send and receive messages.
2. An Accelerator Message Interface for Hardware (AMI-HW) so that hardware accelerators (AHA) can send and receive messages.

The AMI-SW is a device assignable context of a memory-mapped device that allows software to send and receive messages by writing to and reading from ring buffer data structures managed by the AMU.

The AMI-HW is a logical source and destination for messages, associated with a context within an AHA.

This document describes an implementation for AMI-HW as credited streams of messages over a hardware FIFO and a pin-level interface for communicating between the AMU and an AHA. The pin-level interface and implementation for an AMI-HW is not visible to software and is an optional part of this specification.

A2.1.3 Accelerator Session (ASN) and management

An Accelerator Session (ASN) is a unidirectional, ordered stream of messages. Sessions (ASNs) are established to transport messages between a sender and receiver. Privileged software (a Physical Function driver) is responsible for establishment of ASNs and enforcing a security/safety policy by determining which ASNs can be created.

The common case is for a session (ASN) to be from an AMI-SW to an AMI-HW, or an AMI-HW to an AMI-SW. In the case of AMI-SW to AMI-HW, the AMU reads messages from a ring buffer and presents them to an hardware agents (AHA), according to a hardware flow-control scheme. In the case of AMI-HW to AMI-SW, the AMU receives messages from an AHA and writes them to a ring buffer. Sessions (ASNs) between AMI-SW and AMI-HW provide the hardware/software interface for an AHA.

For a session (ASN) created between two AMI-HWs, the AMU forwards messages between hardware agents (AHAs). ASN between AMI-HW are used for chaining of AHA or otherwise autonomous interaction between AHA that does not involve software.

For a session (ASN) created between two AMI-SWs, the AMU copies messages between two ring buffers. Sessions (ASNs) between AMI-SW are typically used for communication of configuration messages between software drivers.

An Accelerator Interface (AMI) is specific to a context with associated state and a view of memory.

An AMI-SW is typically specific to a thread running on a PE.

An AMI-HW is specific to a context in an AHA. In the general case an hardware agent (AHA) supports multiple contexts each with an associated AMI-HW and a PE is switching between multiple threads in multiple exception levels, each thread with a unique AMI-SW.

Note

For use-cases where offload is to software, we expect messages to go via an intermediate AHA such as a DMA engine.

The endpoint of a session (ASN) within an interface (AMI) is referred to as an AMU Message Socket (AMS).

A2.2 Device and I/O Virtualization Model

This section is informative.

The Revere-AMU System Architecture supports a standard device assignment model, that supports virtualization, which is explained in this section. This section should be considered background information and not part of the architecture. This section is included to help explain how system software is expected to use Revere-AMU System Architecture features.

A2.2.1 Background to device assignment and virtualization

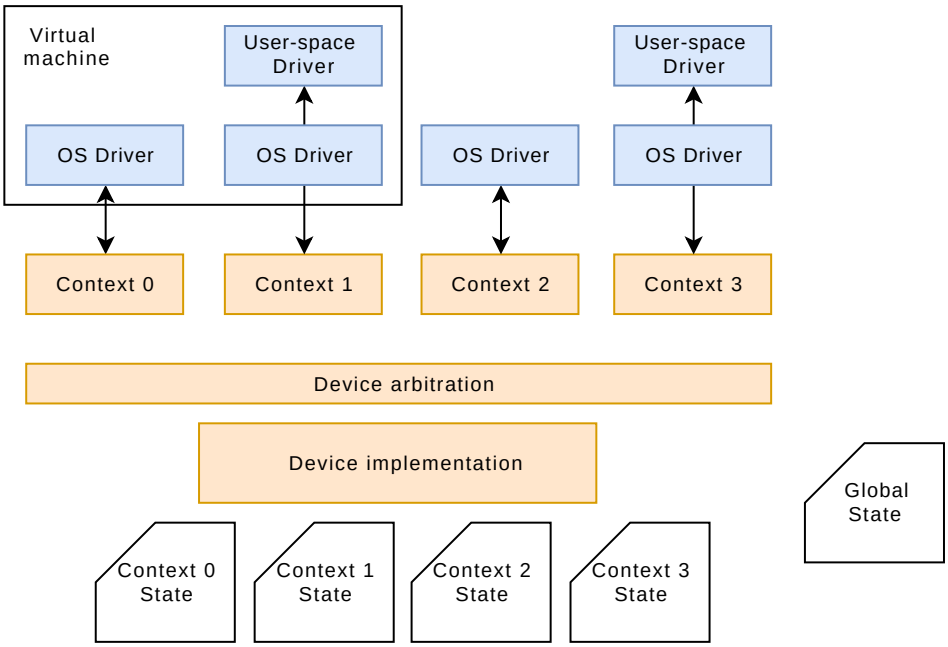

Figure A2.2: Device assignment example showing a device with four contexts. Each context communicates with a driver using memory mapped IO (MMIO) and DMA. A shared device implementation sits behind all contexts and has some global configuration state the affects all contexts. Each context has associated local state maintained in the device implementation.

Devices (I/O or accelerators) support assignment to software within a VM (also known as virtualization pass-through) where the device has one or more contexts (registers, data structures) that can be independently assigned to one or more virtual machines. Device assignment to a VM is also commonly referred to as virtualization pass through. An example implementation of device assignment to a VM is SR-IOV, from the PCI standard. Refer to the PCI Express Base Specification [2].

A device has a common, shared, hardware implementation. For example: a data-path in the case of an accelerator or a physical network port in the case of a network device. This hardware is multiplexed between multiple contexts visible to software.

A device context comprises memory-mapped registers that are accessed by software and/or shared data structures that are accessed by software and by the device. Accesses by software are translated by an MMU. Registers specific to an interface are arranged in the system memory map at a page offset so that the MMU can police access to the interface. Accesses by the device are translated by an SMMU, which allows software to share pointers with the device that reference virtual memory locations.

Each device context must be independent, which means that it must not be possible for software operating on one interface to observe the effect of operations on another interface, except for:

- Where this is desired for correct operation of the device (for example, a network device with multiple interfaces that are all connected using a switch: a send on one can result in a receive event on another).
- Where time multiplexing of hardware may be observable by measuring timing or performance (for example, the device throughput available through one interfaces decreases because of activity on another interface).

To make contexts independent requires replication of state held on the device. The state that needs to be replicated per-interface is device specific.

One context should not be able to block another context, except where this is desired for operation of the device.

Contexts must also have the ability to be reset, so that when it is reassigned to different software, it is in a known state and also leaking of state is prevented.

A device also has global state that is shared between all virtual interfaces. For example a network device might have forwarding rules or a port configuration that affects all interfaces associated with the device.

A common optimization is to replicate only the components needed for high performance per context. For example, each context for a network device only needs to be able to send and receive packets with high performance, which means that only memory-mapped registers and data structures associated with packet send/receive must be replicated per context. The benefit is simplification of the hardware device.

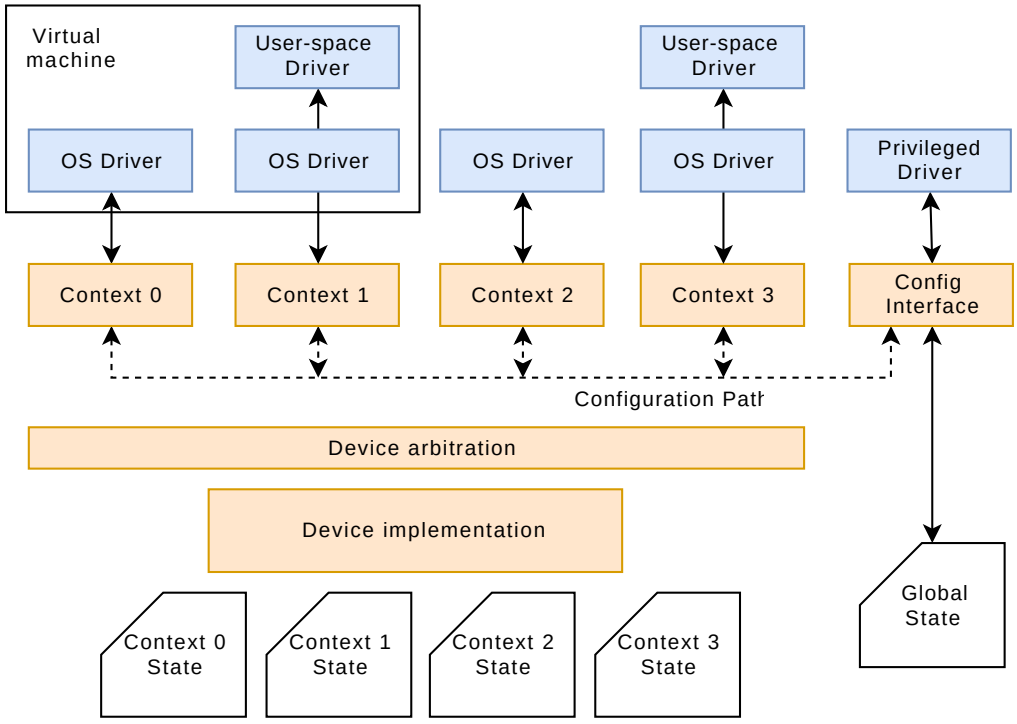

Figure A2.3: Device assignment example showing a device with four contexts. Each context provides a performance path to a shared device implementation and a configuration path to a suitably privileged software driver. The software driver maintains the global configuration state visible to all device contexts.

If each context only replicates the performance path, then the device must provide a separate path for other components of the interface, for example those used for configuration that have a more relaxed performance requirement. A common approach shown in Figure 3 is to delegate management of configuration to a software driver running in an appropriately privileged state. This driver controls all device configurations through a suitable interface and has a messaging path to each of the software drivers using the device. The benefit (in addition to simplification of the device, as described above) is that potentially complex arbitration rules, that may be essential

for security and safety of the system, can be implemented in a software driver. If system software supports a paravirtualised path between drivers, or all required controls are replicated per context then the messaging path through the device is not used and can be disabled.

A2.2.2 Device assignment in the Revere-AMU System Architecture

In the Revere-AMU architecture, a device is a combination of a single AMU and zero or more AHA (AMU Hardware Agents) that participate in sending and receiving messages using that AMU. The AHAs implement the shared, application specific data-path of a particular device and the AMU implements a generic interface to software that supports device assignment.

The AMU represents a device as multiple separate interfaces to software. This specification uses the PCI SR-IOV terminology to refer to software that uses the AMU interfaces.

A single PF (Physical Function) driver in the system controls the global configuration of the AMU and also any AHA (Accelerator Hardware Agent) associated with the AMU.

Multiple VF (Virtual Function) drivers in the system each control the configuration of the AMU relating to a single virtual machine. The VF driver is responsible for controlling device assignment within a virtual machine. The VF driver can be generic (able to control assignment without any knowledge of the particular device) or specific. A specific VF driver can incorporate the user driver.

In addition to managing the global configuration of the AMU (which is PF specific), the PF driver is also equivalent to a VF driver for the purposes of I/O and device assignment. This part of the PF driver can be generic part.

This document refers to Functions and Function drivers when a particular behaviour applies to both, PF and VF.

Multiple User drivers in the system send and receive messages only. The user driver is the software that actually uses the I/O or accelerator managed by the AMU and is device specific.

Table A2.1: Summary of software that uses interfaces provided by the AMU

Software	Number in system	Privilege level	AMU interfaces	Responsibilities
PF driver	1	>= VF driver	Management Interface & AMI-SW	Global configuration and send/receive messages
VF driver	0-N	>= User driver	Management Interface & AMI-SW	Configuration of a VF and send/receive messages
User driver	0-N		AMI-SW	Send/receive messages

The AMU presents Accelerator Message Interfaces for software (AMI-SW). Each AMI-SW is independently assignable to a software driver. An AMI-SW can provide the high-performance path for the driver to communicate with another context. This includes a context running on a hardware agent (offload to an accelerator) or a context running on another PE (offload to software).

The AMU also presents a Management Interface per PCI Express Function. The Management Interface includes a small set of management registers and one management AMI, allowing the discovery and configuration path for both the PF and VF drivers to send configuration management requests to the AMU, and for the VF driver to send configuration management requests to the PF driver, used to manage configuration state.

The configuration includes allocation of resources across the system (for example, the AMI-SW available to each software entity) and setting up of ASNs (Accelerator Sessions) (for example, connecting a device driver thread to a corresponding AMI on an AHA).

Each hardware agent (AHA) that participates in sending/receiving messages using the AMU optionally presents a configuration interface as memory mapped registers (outside the scope of this architecture) which the PF Driver uses to manage configuration specific to that AHA implementation.

The single PF driver, that is specific to the device, is responsible for managing the device configuration as well as the AMU configuration. A typical life cycle of a PF driver is:

1. Perform device specific initialization.
2. Configure the AMU, apportioning software visible AMI-SWs across the system. This setup depends on the software make-up of the system (for example: many VMs vs many containers) and the needs of the device (for example: the number of sessions needed per logical interface to the device).
3. Set up needed sessions (ASNs) from the AMI-SWs visible to software and corresponding AMIs on the device.
4. (Optionally) Receive device specific configuration messages from a VF driver and use to modify the global state of the device.
5. (Optionally) Dynamically set up/tear down ASNs in response to device configuration messages. The VF driver is responsible for managing the AMU configuration for a single virtual machine.

A typical life cycle of a VF driver is:

1. Configure the assignment of AMI-SW to a user driver.
2. (Optionally) Dynamically reassign AMI-SW between user drivers.

The mechanisms that the user driver can use to interact with the device are intentionally limited to:

- Sending a message to the device (subject to privileged software setting up an appropriate session)
- Receiving a message from a device (subject to privileged software setting up an appropriate session)
- Reading/writing buffers in the virtual address space of software that are referenced by a message (including other memory operations that may be supported by the underlying memory system, such as atomics).
- Making a system call to the VF driver, which then sends a device-specific configuration message to the device-specific PF driver. The PF driver then modifies a device-specific set of memory mapped registers or data structures.

It is the responsibility of Function drivers and other system software to modify page tables and/or the SMMU configuration in order to restrict DMA operations by the AMU or devices. Revere-AMU messages simply provide a convenient abstraction for linking memory operations to stream/substream ids for the SMMU.

Some operating systems/hypervisors provide a software (para-virtualized) mechanism between privileged and non-privileged drivers, in which case it is not necessary to use the AMU to exchange configuration messages between the PF and VF drivers. The PF driver can disable the configuration messaging feature in this instance.

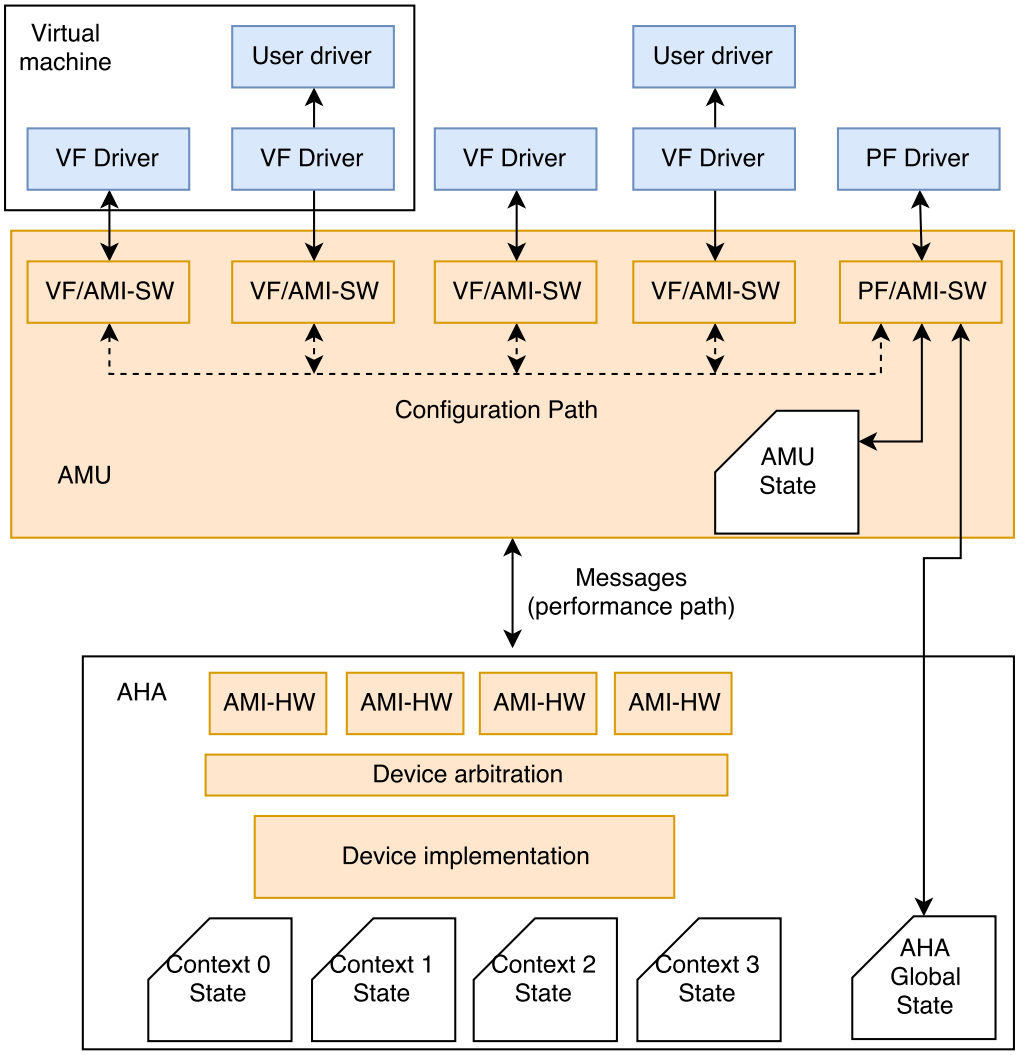

Figure A2.4: Device virtualization example in the Revere-AMU System Architecture. In this example, the AMU provides a performance path and configuration path for four virtual interfaces (AMI-SW). The performance path and configuration path are abstracted as sending and receiving messages through the AMU.

A2.3 System Placement

The AMU has a management interface, exposed to software through the PF/VF Memory Space aperture defined by its BAR0, for:

- Enumeration, configuration and maintenance.
- Sending and receiving of messages by software.

In addition, the AMU initiates its own accesses for the purpose of reading/writing data structures shared with software. These accesses must be translated by an SMMU.

An AMU is associated with zero or more Accelerator Hardware Agents (AHA) that also send and receive messages. These AHAs optionally initiate direct memory accesses. Direct memory accesses from AHAs must also be translated by an SMMU. These AHAs optionally receive accesses from software for configuration.

Each AMU and all associated AHAs must use the same SMMU to provide translations.

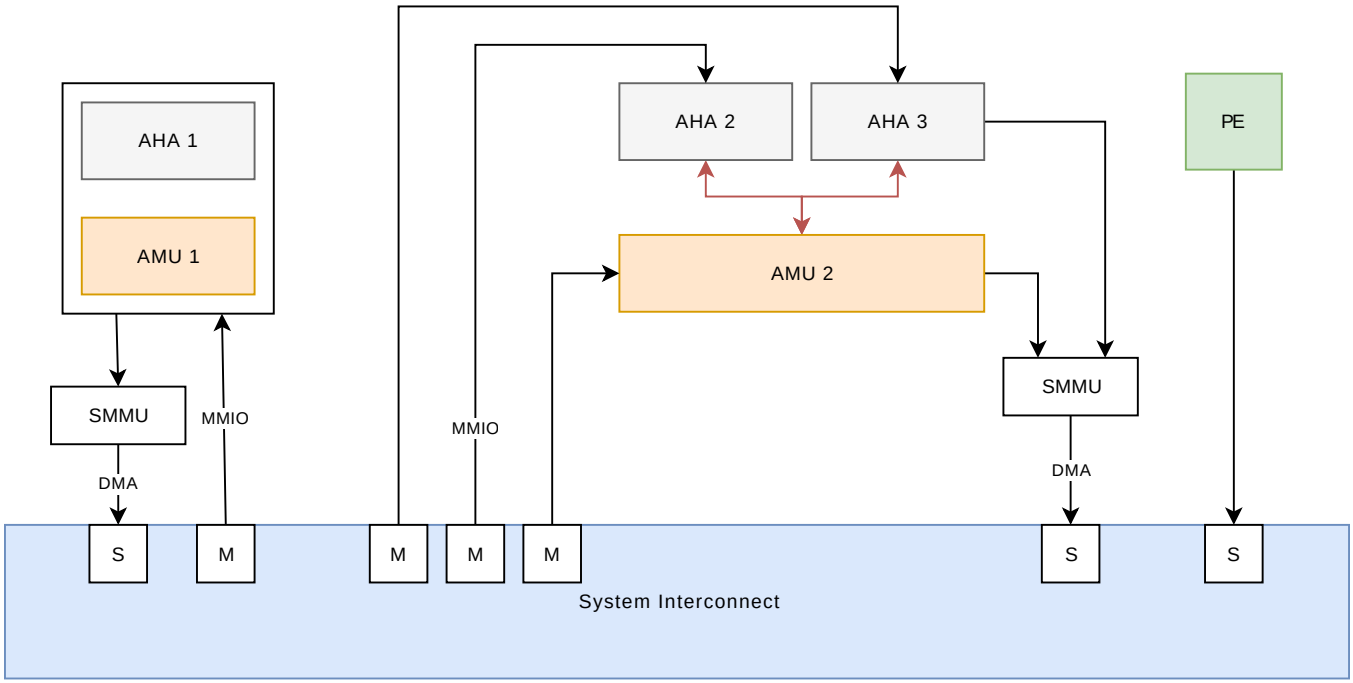

Figure A2.5: Example system with two AMUs. AMU1 is integrated in the same functional block as a single AHA. AMU2 is implemented as a separate functional block from AHAs 2 and 3. An IMPLEMENTATION DEFINED pin-level interface connects AHA2 and AHA3 to AMU2. AHA3 has a separate DMA master interface into the system and must use the same SMMU as AMU2 for translations. Messages can be exchanged between software running on any PE in the system and any of the AHAs. AHA2 and AHA3 can exchange messages. AHA1 cannot exchange messages with AHA2 or AHA3

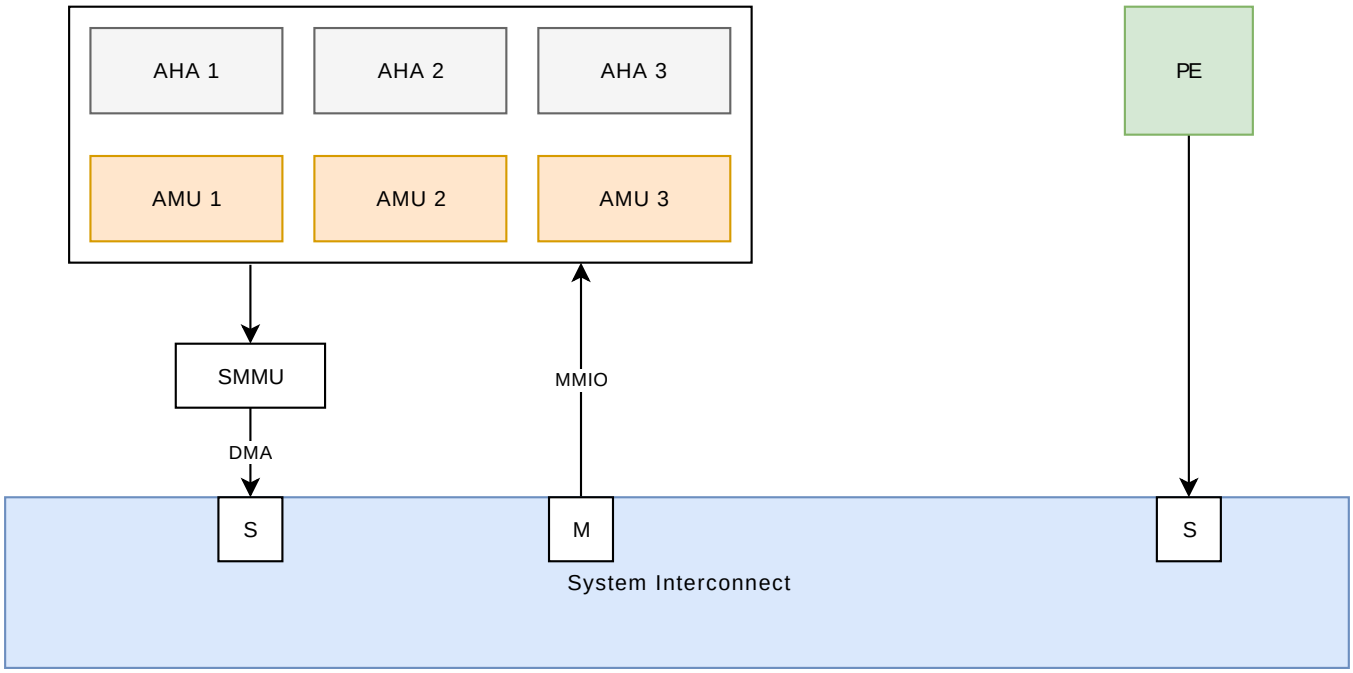

Figure A2.6: Example system with three AMUs. Each AMU has a single AHA and all are implemented in the same functional block. Messages can be exchanged between software running on any PE in the system and any of the AHAs. No AHA can exchange messages with another AHA.

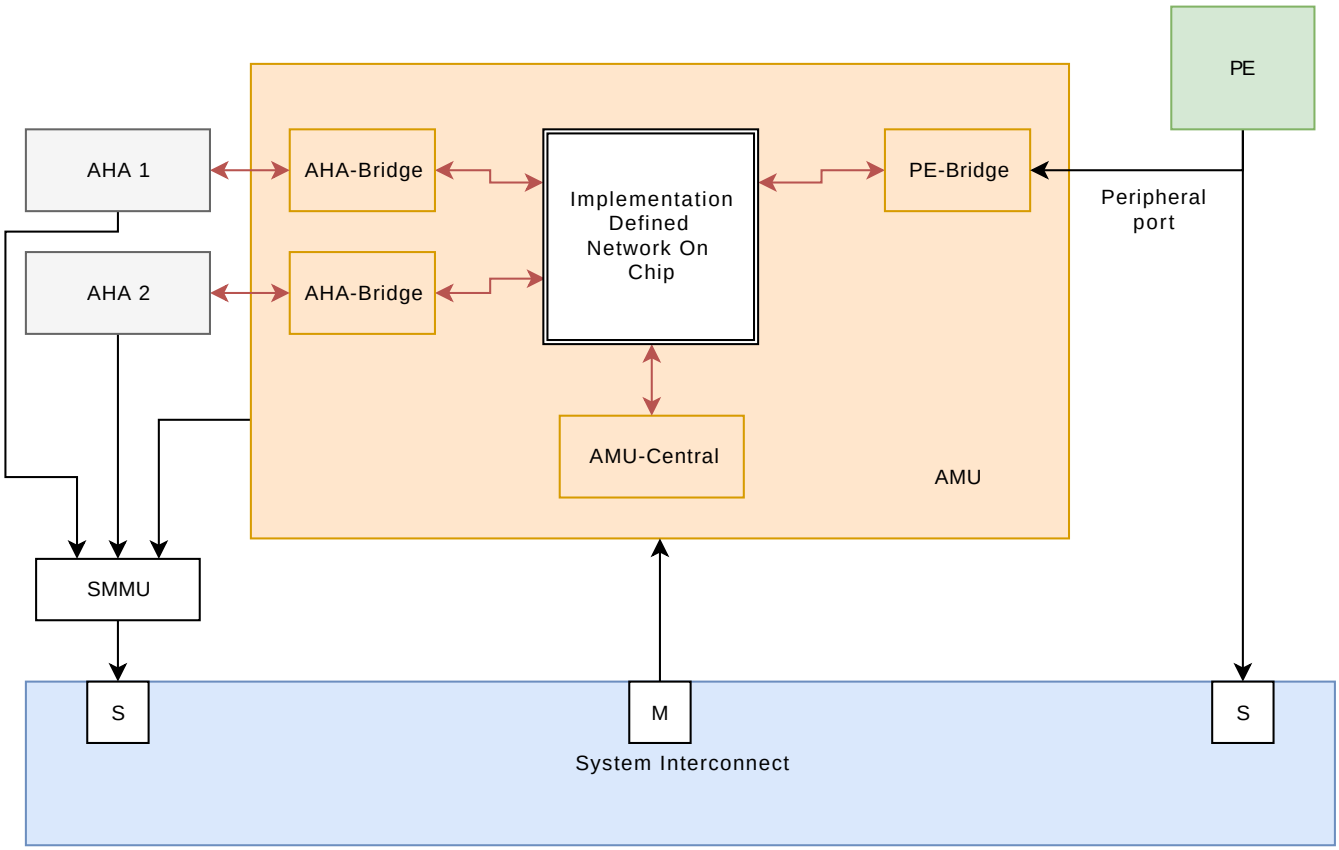

Figure A2.7: Example system with a single AMU. The AMU is disaggregated into multiple functional blocks using an IMPLEMENTATION DEFINED network on chip. AHA1 and AHA2 both have separate DMA master interfaces and must use the same SMMU as the AMU for translations. The AMU implements certain registers relating to an AMI-SW local to a PE in an IMPLEMENTATION DEFINED way.

The implementation of an AMU may be centralised or distributed.

An AMU might be contained in a single functional block with zero or more hardware agents.

An AMU itself may be physically disaggregated, for example to distribute the management of AMI-SW close to the PE running the software thread. How to disaggregate the AMU in this way is outside the scope of the architecture.

An AMU may be a separate functional block from the one or more hardware agents that send/receive messages. This specification provides an optional pin-level interface between the AMU and associated hardware agents.

Multiple AMUs may be implemented using the same functional block, or the same set of physically disaggregated functional blocks. How to use the same functional blocks to implement multiple AMUs is outside the scope of the architecture.

An AMU represents a device that is independently managed by system software. The number of AMUs in the system and the number of AHAs per AMU depends on the nature of the device and how it needs to be viewed by system software.

Hardware agents (AHAs) that need to exchange messages directly (for example, to chain operations) must be managed by a common AMU.

AHAs that are unrelated, and need to be managed separately by system software (separate Physical Function drivers), should be managed by different AMUs.

Separate AHAs may share resources, or be implemented as a single functional block.

Chapter A3
Architecture

This chapter provides a description of Revere-AMU System Architecture.

A3.1 Overview

A3.1.1 Agents and contexts

In the Revere-AMU System Architecture, a system is composed of AMU Hardware Agents (AHAs). Examples of AHAs include an accelerator and an I/O device.

Hardware agents (AHAs) optionally initiate transactions to a memory system that is shared and coherent between all other hardware agents (AHAs) .

AHAs perform work within a context. A context may include state. If the AHA initiates transactions to memory, the context includes a view of memory via a Memory Management Unit (MMU) or a System MMU (SMMU) for devices.

How an AHA supports multiple contexts is IMPLEMENTATION DEFINED. Support for multiple contexts in an AHA is analogous to software threads running on a CPU, for example the Arm v8 Architecture Reference Manual describes how an Arm v8 PE supports switching between multiple contexts.

An Accelerator Management Unit (AMU) manages sending and receiving of messages between contexts within the AHAs in the system associated with that AMU.

Note

A system can include multiple AMUs. AHAs are each associated with exactly one AMU. It is not possible to send messages between AHAs associated with different AMUs.

AMU Hardware Agents (AHAs) that send/receive messages via an AMU and initiate transactions to the memory system must use an SMMU. Each AMU and all associated AHAs must use the same SMMU to provide translations.

Note

It is permissible to have multiple SMMUs in the system as long as there is not more than one SMMU in use by AHAs associated with the same AMU.

A3.1.4 *Messages* describes the abstract definition of a message format (that applies to both AMI-SW and AMI-HW).

A3.1.2 Accelerator Message Interface (AMI)

Each AMU Hardware Agent (AHA) supports a defined number of Accelerator Message Interfaces (AMIs). Each context uses an AMI for the purpose of sending and receiving messages. AMIs are logical concepts: an AHA that supports multiple contexts must also support multiple AMIs (one per context) using the same physical hardware resources.

Table A3.1: Interfaces/contexts and how they apply to an Arm v8 PE and AMU Hardware Agents (AHAs). Physical implementations shown are examples

	Arm v8 PE uses AMI-SW	**AHA uses AMI-HW**
Logical (Interface)	AMI-SW assignable context of memory mapped device described in A3.7 *Accelerator Message Interface for Software*.	AMI-HW message stream with specified AMI id.
Context	View of memory. PE architected state.	View of memory. AHA specific state.

	Arm v8 PE uses AMI-SW	AHA uses AMI-HW
Physical implementation (example, not in the architecture scope)	CPU master port to interconnect for read/write of data structures and MMIO registers.	Interconnect for sending/receiving messages.

Since there is a 1:1 correspondence of AMIs to contexts, the AMI shares the view of memory and any state maintained within that context.

A3.1.4 *Messages* describes the AMI-SW for an Arm v8 PE to send and receive messages using Revere-AMU. Each software context has direct access to a memory mapped logical interface (AMI) via shared data structures and memory mapped I/O reads/writes. The logical AMI is defined by the address of those data structures and memory mapped I/O registers. System software ensures through standard virtual memory protection mechanisms that each software context has unique access to the addresses associated with an interface.

An AMI-SW can cause the generation of interrupts toward a PE to notify of events, such as the reception of a message. This is detailed in A3.7.7 *Interrupts*.

Chapter B2 *AMU AHA Interface protocol* describes an optional implementation of the AMI-HW using a pin-level interface for an AHA implemented as a separate functional block to send and receive message using a Revere-AMU implemented as a different functional block.

AMU Interfaces (AMIs) are configured through the AMU management interface (see A3.4.4 *Management Messages*).

There are up to MAX_AMI_SW software AMI in an AMU.

A3.1.3 Accelerator Message Socket (AMS) and Accelerator Session (ASN)

Each interface (AMI) has zero or more receive sockets (AMSs) for receiving messages and zero or more transmit sockets (AMSs) for sending messages.

Each AMS is associated with exactly one AMI.

A session (ASN) is a connection between exactly one TX socket (AMS) and one RX socket (AMS) in two different interfaces (AMIs), for the purpose of exchanging messages. Messages are ordered within a session (ASN).

Messages can be buffered at the TX socket (AMS), the depth of the buffering is IMPLEMENTATION DEFINED. Messages are credited end to end between the RX and TX sockets (AMS). The RX AMS must provide buffering for one or more messages. Sockets (AMSs) within an AMI are identified by a AMS ID. AMSs are uniquely identified within an AMU by an AMI ID and a AMS ID.

Sessions (ASNs) are uniquely identified within an AMU by a ASN ID.

A3.1.4 Messages

The AMU exchanges messages between contexts running on hardware agents (AHAs) and on Armv8 PEs. The high-level structure of the message is described in Figure A3.1.

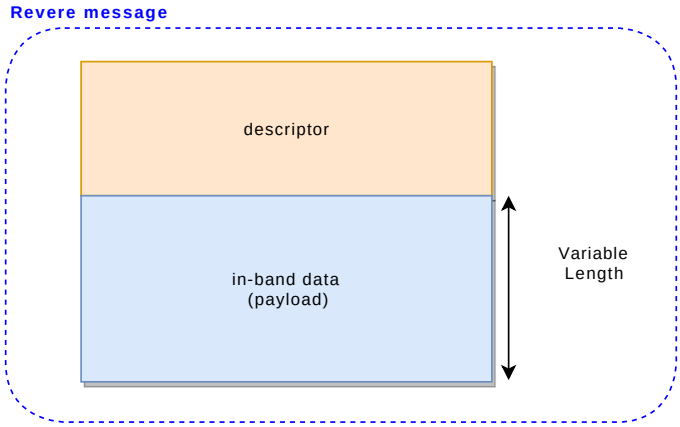

Figure A3.1: Revere-AMU Message.

Messages contain a descriptor and a variable size payload (referred as in-band data). Revere-AMU messages might also contain one or multiple pointers to associated buffers in memory address space referred as out-of-band data buffers. A3.2 *Message format* provides a complete description of all the supported message formats.

The AMU transports messages between the source and destination sockets (AMSs). While some use cases might not require the movement of the associated out-of-band data, others require the out-of-band buffers to be moved. Typical use cases requiring data movements are listed here:

- VM to VM communication
- SW agent memory to a slave hardware agent (AHA)
- Slave AHA to a SW agent memory

The AMU is not responsible for moving out-of-band data buffers. Instead, it is the responsibility of hardware agents (AHAs) with DMA capabilities.

Note

It should be noted that messages do not contain any destination information. The routing information is defined by the ASN.

A3.1.5 Endianness

All Revere data structures are in little-endian format.

A3.2 Message format

As described in section A3.1.4 *Messages*, Revere-AMU messages can have various message format options (MFO).

A message includes a descriptor and a payload.

Some message formats reference one or more out-of-band buffers. In that case, the descriptor contains information (such as addresses in memory, sizes, cache stashing control information (A3.5 *Cache stashing*), etc.) about the out-of-band buffers.

An AHA is allowed to generate DMA accesses to any virtual address from any source pointer.

The difference with architected pointers accesses is that implementations might not be able to act (issuing cache stashing hint requests for example) on data referenced by a pointer that is not in an architected field.

The system software can always enable or restrict those accesses by configuring the SMMU.

The Message Format Option (MFO) is a property of a session and is configured through ASN configuration messages A4.2.5 *ASN Management Commands and Responses*.

All messages exchanged over a given session (ASN) have the same MFO.

The session parameter LOG2_MSG_LENGTH (See A4.2.5 *ASN Management Commands and Responses*) determines the size of all the messages exchanged over a given ASN.

- In number of Doublewords and expressed as $\log_2$(MSG_LENGTH).
- The maximum message size is 512 Doublewords = 4KB.
- Maximum value for LOG2_MSG_LENGTH is 9.

The AMU uses this parameter to select a source and destination buffering size.

- In the case of an AMI-SW: size of a slot of the ring buffer slot array.
- In the case of an AMI-HW: size of the AHA internal buffering.

A3.2.1 Message Format Option 0

Message Format Option (MFO) 0 (Figure A3.2) provides the simplest message and only contains a payload. This message format would typically be used for short commands and acknowledge messages.

MFO0 messages size is configured with the parameter LOG2_MSG_LENGTH set during the session creation (PF-ASN-CREATE).

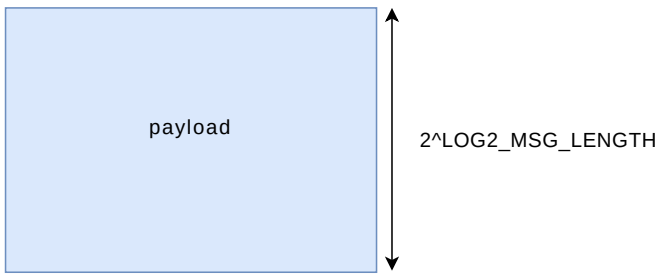

Figure A3.2: MFO0 Message structure

For MFO0, stashing of the message data is controlled by the session STASH_CTL parameter.

A3.2.2 Message Format Option 1

Message Format Option (MFO) 1 (Figure A3.3) is similar to MFO0 but provides a variable size defined - in the descriptor of the message - with the LENGTH parameter.

LENGTH defines the number of Doublewords of the MFO1 message (descriptor + payload).

LENGTH shall be smaller or equal to $2^{LOG2\_MSG\_LENGTH}$. When LENGTH is larger than $2^{LOG2\_MSG\_LENGTH}$, only the first $2^{LOG2\_MSG\_LENGTH}$ Doublewords are transferred.

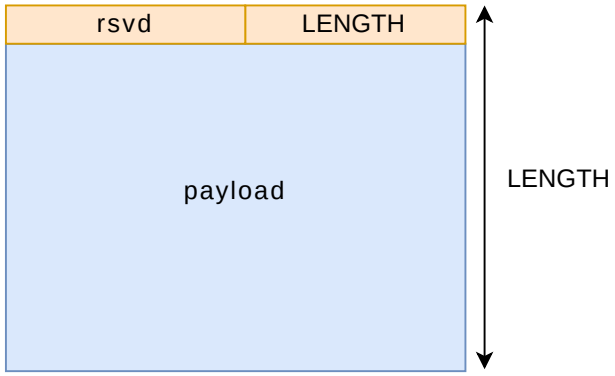

Figure A3.3: MFO1 Message structure

Table A3.2: Descriptor for MFO1

Offset	Bits	Name	Meaning
+0x00	[63:9]	- Reserved -	Reserved, RES0
+0x00	[8:0]	LENGTH	Size of the message (descriptor + payload) in number of Doublewords (minus one)

For MFO1, stashing of the message data is controlled by the session STASH_CTL parameter.

A3.2.3 Message Format Option 2

Message Format Option (MFO) 2 (Figure A3.4) provides references to a set of equal size out-of-band buffers.

MFO2 messages size is configured with the parameter LOG2_MSG_LENGTH set during the session creation (PF-ASN-CREATE).

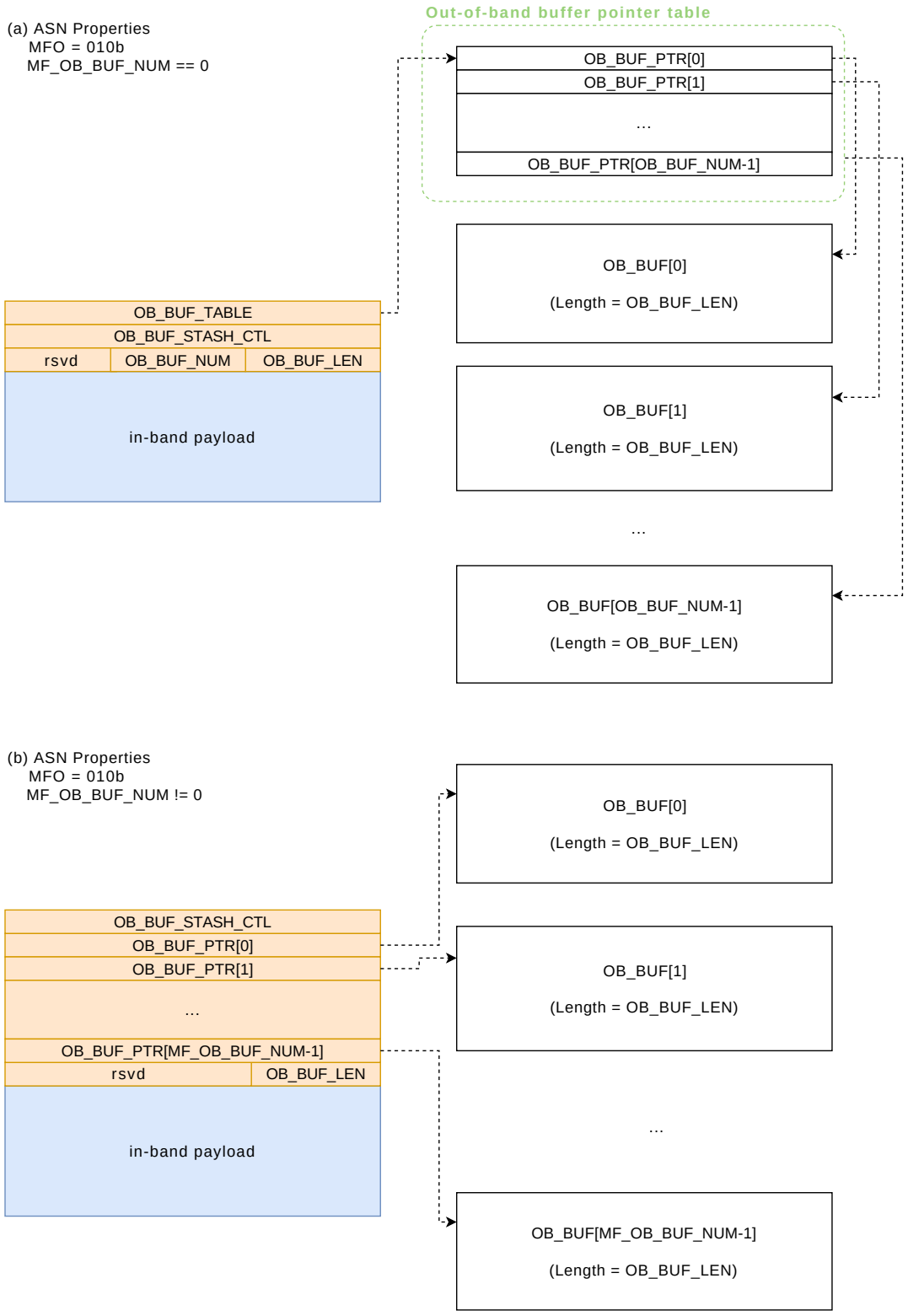

Figure A3.4: MFO2 Message structure

All out-of-band buffers have the same size (defined by the `OB_BUF_LEN` field in the descriptor) and the same cache stashing control information (defined by the `OB_BUF_STASH_CTL` field in the descriptor).

The references can either be embedded in the message descriptor or stored in an out-of-band buffer table in Normal memory.

The session parameter MF_OB_BUF_NUM (See A4.2.5 *ASN Management Commands and Responses*) determines the number of out-of-band buffer references embedded in the descriptor.

MF_OB_BUF_NUM == 0 indicates that the out-of-band buffer references are stored in an out-of-band buffer table. The descriptor just contains a reference to the out-of-band buffer table.

Table A3.3: Descriptor for MFO2 when MF_OB_BUF_NUM == 0

Offset	Bits	Name	Meaning
+0x00	[63:0]	OB_BUF_TABLE	Pointer to the out-of-band buffer table
+0x08	[63:0]	OB_BUF_STASH_CTL	Stashing control for the out-of-band buffers
+0x10	[63:29]	- Reserved -	Reserved, RES0
+0x10	[28:22]	OB_BUF_NUM	Number of out-of-band buffers
+0x10	[21:0]	OB_BUF_LEN	Length of the out-of-band buffers in bytes

Revere's message includes a pointer to an out-of-band buffer table that contains `OB_BUF_NUM` entries. Table A3.4 describes the format of one entry.

Table A3.4: Out-of-band buffer table entry format for MFO2

Offset	Bits	Name	Description
+0x00	[63:0]	OB_BUF_PTR	Pointer to the out-of-band buffer table

When MF_OB_BUF_NUM != 0, MF_OB_BUF_NUM out-of-band buffer pointers are directly embedded in the descriptor.

N= MF_OB_BUF_NUM

Table A3.5: Descriptor for MFO2 when MF_OB_BUF_NUM != 0

Offset	Bits	Name	Meaning
+0x00	[63:0]	OB_BUF_STASH_CTL	Stashing control for the out-of-band buffers
+0x08	[63:0]	OB_BUF_PTR[0]	Pointer to out-of-band buffer #0
+0x08 + 8	[63:0]	OB_BUF_PTR[1]	Pointer to out-of-band buffer #1
…	…	…	…
+0x08 + 8*(N-1)	[63:0]	OB_BUF_PTR[N-1]	Pointer to out-of-band buffer #[N-1]
+0x08 + 8*N	[63:22]	- Reserved -	Reserved, RES0
+0x08 + 8*N	[21:0]	OB_BUF_LEN	Length of the out-of-band buffers in bytes

For MFO2, stashing is controlled in the following way:

- Stashing of the message data is controlled by the session STASH_CTL parameter
- When MF_OB_BUF_NUM is equal to zero, stashing of the out-of-band buffer table is controlled by the session STASH_CTL parameter, with STASH_OFFSET and STASH_LEN fields ignored
- Stashing of the out-of-band buffers is controlled by the message descriptor OB_BUF_STASH_CTL

A3.2.4 Message Format Option 3

Message Format Option (MFO) 3 (Figure A3.5) provides references to a set of out-of-band buffers.

Each buffer can be of different size and can have its own cache stashing control information.

MFO3 messages size is configured with the parameter LOG2_MSG_LENGTH set during the session creation (PF-ASN-CREATE).

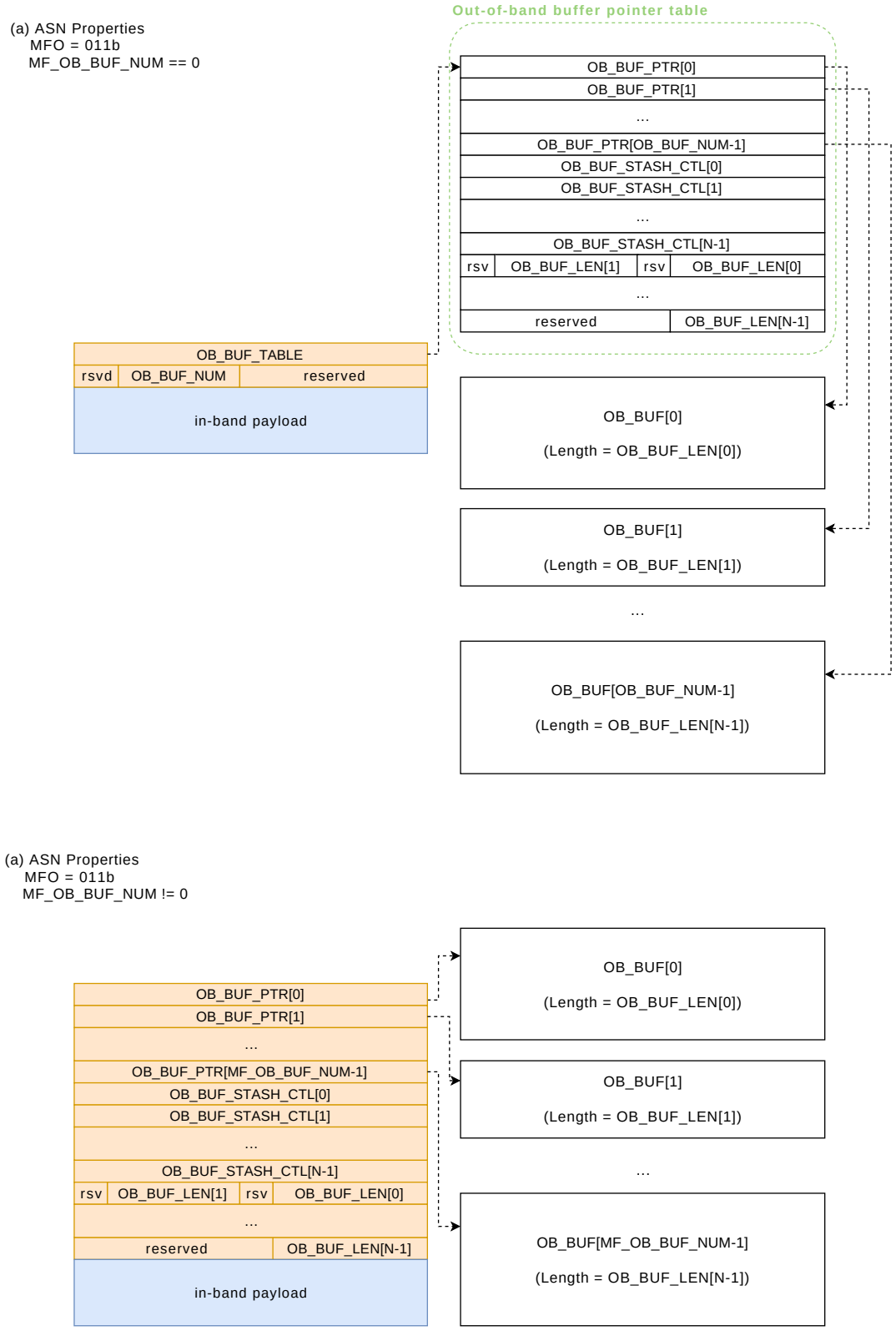

Figure A3.5: MFO3 Message structure

The session parameter MF_OB_BUF_NUM (See A4.2.5 *ASN Management Commands and Responses*) determines

the number of out-of-band buffer references embedded in the descriptor.

MF_OB_BUF_NUM == 0 indicates the parameters defining the lengths of the out-of-band buffers and the cache stashing control information are not located in the descriptor. Instead, they are located in the out-of-band buffer table.

Table A3.6: Descriptor for MFO3 when MF_OB_BUF_NUM == 0

Offset	Bits	Name	Meaning
+0x00	[63:0]	OB_BUF_TABLE	Pointer to the out-of-band buffer table
+0x08	[63:29]	- Reserved -	Reserved, RES0
+0x08	[28:22]	OB_BUF_NUM	Number of out-of-band buffers
+0x08	[21:0]	- Reserved -	Reserved, RES0

Revere's message includes a pointer to an out-of-band buffer table. Table A3.7 describes the layout of this table:

N= OB_BUF_NUM

Table A3.7: Out-of-band buffer table entry format for MFO3

Offset	Bits	Name	Description
+0x00	[63:0]	OB_BUF_PTR[0]	Pointer to out-of-band buffer #0
+0x08	[63:0]	OB_BUF_PTR[1]	Pointer to out-of-band buffer #1
...	...	...	...
+0x00 + 8*(N-1)	[63:0]	OB_BUF_PTR[N-1]	Pointer to out-of-band buffer
+0x08 + 8*(N-1)	[63:0]	OB_BUF_STASH_CTL[0]	Stashing control information
+0x10 + 8*(N-1)	[63:0]	OB_BUF_STASH_CTL[1]	Stashing control information
...	...	...	...
+0x08 + 2*8*(N-1)	[63:0]	OB_BUF_STASH_CTL[N-1]	Stashing control information
+0x10 + 2*8*(N-1)	[31:22]	- Reserved -	Reserved, RES0
+0x10 + 2*8*(N-1)	[21:0]	OB_BUF_LEN[0]	Length of buffer #0 in bytes.
+0x14 + 2*8*(N-1)	[31:22]	- Reserved -	Reserved, RES0
+0x14 + 2*8*(N-1)	[21:0]	OB_BUF_LEN[1]	Length of buffer #1 in bytes.
...	...	...	...
+0x10 + 2*8*(N-1) + 4*(N-1)	[31:22]	- Reserved -	Reserved, RES0

Offset	Bits	Name	Description
+0x10 + 2*8*(N-1) + 4*(N-1)	[21:0]	OB_BUF_LEN[N-1]	Length of buffer[N-1] in bytes.

When MF_OB_BUF_NUM != 0, MF_OB_BUF_NUM out-of-band buffer pointers are directly embedded in the descriptor. In that case, the layout of the descriptor is similar to the out-of-band buffer table (Table A3.7) with OB_BUF_NUM replaced with MF_OB_BUF_NUM.

For MFO3, stashing is controlled in the following way:

- Stashing of the message data is controlled by the session STASH_CTL parameter
- When MF_OB_BUF_NUM is equal to zero, stashing of the out-of-band buffer table is controlled by the session STASH_CTL parameter, with STASH_OFFSET and STASH_LEN fields ignored
- Stashing of the out-of-band buffers is controlled by the out-of-band buffer table or message descriptor OB_BUF_[STASH_CTL][n]

A3.2.5 Message Format Option 4

When using Message Format Option (MFO) 4 (Figure A3.6), the descriptor contains a pointer to beginning of a linked list of `OB_BUF_NUM` out-of-band buffers. This format enables to associate non-equal size buffers. For this message format, Revere System Architecture defines a header that has to be prepended to each out-of-band buffer in order to describe the length of out-of-band buffer, the cache stashing characteristics and a pointer to next out-of-band buffer in the linked list.

A NULL pointer specifies the end of the linked list.

MFO4 messages size is configured with the parameter LOG2_MSG_LENGTH set during the session creation (PF-ASN-CREATE).

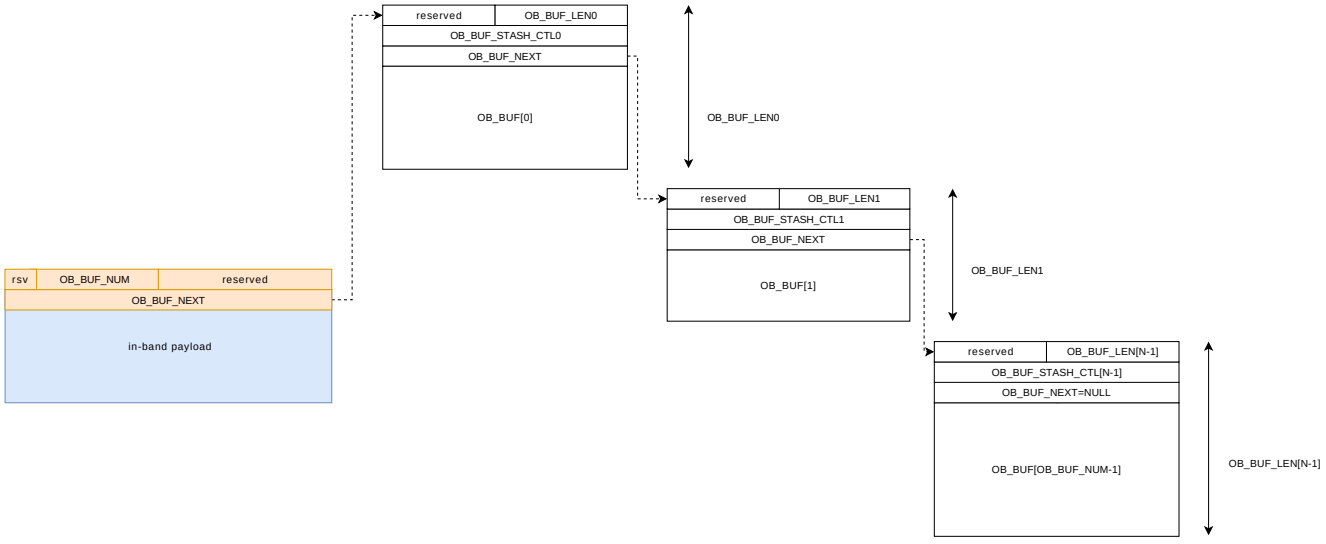

Figure A3.6: MFO4 Message structure

Table A3.8: Descriptor for MFO4

Offset	Bits	Name	Meaning

Offset	Bits	Name	Meaning
+0x00	[63:29]	- Reserved -	Reserved, RES0
+0x00	[28:22]	OB_BUF_NUM	Number of the out-of-band buffers
+0x00	[21:0]	- Reserved -	Reserved, RES0
+0x08	[63:0]	OB_BUF_NEXT	Pointer to the first out-of-band buffer

For MFO4, stashing is controlled in the following way:

- Stashing of the message data is controlled by the session STASH_CTL parameter
- Stashing of each out-of-band buffer is controlled by the OB_BUF_STASH_CTL in its header

See A3.5 *Cache stashing*.

A3.2.5.1 Out-of-band buffer header for MFO4

Revere's message includes a pointer to the first out-of-band buffer of a chain of `OB_BUF_NUM` buffers. Table A3.9 describes the format of the header prepended to each out-of-band buffer.

Table A3.9: out-of-band buffer header format for MFO4

Offset	Bits	Name	Description
+0x00	[63:22]	- Reserved -	Reserved, RES0
+0x00	[21:0]	OB_BUF_LEN	length of the buffer in bytes
+0x08	[63:0]	OB_BUF_STASH_CTL	Stashing control information for the out-of-band buffer
+0x10	[63:0]	OB_BUF_NEXT	Pointer to next entry

A3.3 Revere-AMU and PCI Express

A3.3.1 Overview

The Revere-AMU memory mapped device exposes to software:

- A single Physical Function, used to control the global configuration of the device.
- Zero or more Virtual Functions, used to control the configuration local to a single virtual machine.
- A set of assignable device contexts, called AMI-SW, used for sending and receiving messages.

Functions and AMI-SWs have separate controls for translation scheme and the location/discovery. Each Function and AMI-SW has independent controls over the location of memory mapped registers in the system address space and the translation scheme used for DMA.

Each Function contains one management interface to exchange configuration messages with the AMU (see A3.4 *Management Interface*).

AMI-SWs are mapped to Functions by the PF driver through the management interface. Mapping an AMI-SW to a Function affects the stage 2 translation scheme used for DMA by that AMI-SW and the location of its associated memory mapped registers.

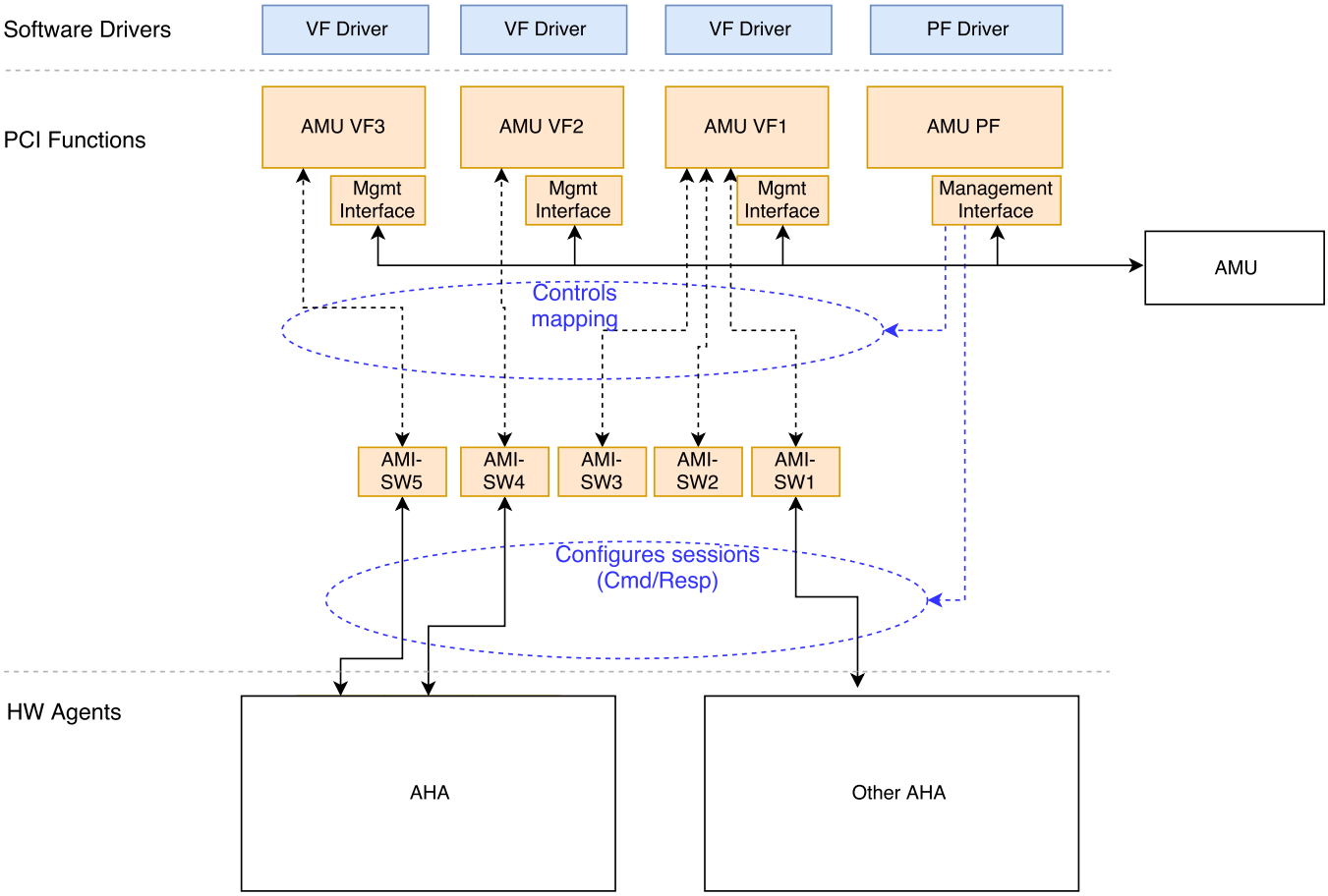

Figure A3.7: Functions contain a management interface and a configurable set of AMI-SWs. AMI-SWs are mapped to Functions by the PF driver. Configuring ASNs between AMSs is also the responsibility of the PF driver.

The PF and VF drivers use their management AMI to:

- Exchange architected message types that manage the AMU.
- Exchange configuration messages in the system between PF and VF drivers.
- Receive traces from the AMU, if this feature is implemented.
- Receive notifications of asynchronous exceptions.

The AMU optionally acts as a 'message switch' for configuration messages in the system. The AMU switching capability can be disabled by software when system software provides a para-virtualized path for configuration messages (see A3.4.3.4 *AMU_CR*).

Each AMI-SW:

- Has multiple TX and RX sockets (AMSs), the number of each is fixed by the implementation and discoverable by software.
- Has a DMA capability for reading/writing data structures associated with those AMSs.
- Has memory mapped registers for discovering capabilities associated with those AMSs.
- Has memory mapped registers for maintaining digests (a summary of the state of multiple sockets) and to control interrupts.
- Is optionally associated with a stage 1 translation scheme, and optionally a stage 2 translation scheme from the owning Function.
- Can be independently reset.
- Can optionally generate an interrupt vector associated with TX socket (AMSs) events, and an interrupt vector associated with RX sockets (AMSs) events.

Each Function is a granule of AMI-SWs that can be assigned by system software to a virtual machine. Each Function:

- Has controls for the stage 1 translation scheme for each associated AMI-SW.
- Is associated with a stage 2 (and optionally also a stage 1) translation scheme.
- May have multiple AMI-SWs.
- Can optionally have an MSI-X Capability, allowing to generate and configure interrupts.
- If supported, has independent tracing capabilities.
- If supported, has independent profiling capabilities.

A3.3.2 PF/VF Configuration Space

The AMU is visible to software as PCI Functions within a PCI Express Endpoint as defined by the PCI Express Base Specification [2].

For an on-chip implementation, the AMU is visible to software as a Root Complex Integrated Endpoint. In the on-chip case, all Functions should follow the guidelines for a Root Complex Integrated Endpoint as per chapter "Root Complex Integrated Endpoint Rules" in the PCI Express Base Specification [2]. The AMU implements a single Physical Function (PF) and zero or more Virtual Functions (VF).

All Functions implement a Type 00h PCI compatible configuration space and all architecturally required PCI Express capabilities.

All Functions implement all software visible features required by a PCI Express Function. All Functions implement the behaviour that is architecturally required by the PCI Express Base Specification [2].

All Revere-AMU Virtual Functions must implement the same set of capabilities.

All DMA behind a Function must implement the correct behaviour for the Bus Master Enable bit and Outstanding Request status for that Function. This includes:

- DMA within the AMI-SW that accesses ring buffers and other data structures.
- DMA that is part of other hardware agent (AHA) that accesses buffers referenced in messages.
- DMA performing a write transaction, to generate a Message Signalled Interrupt.

Refer to Table A3.45 for details on DMA accesses.

A Revere-AMU device implements the following PCI-Compatible Configuration Registers and PCI Express Capabilities:

	Physical Function	Virtual Functions
Type 0 Configuration Space	Required	Required
PCI Power Management Capability	Required	Optional
PCI Express Capability	Required	Required
MSI-X Capability	Optional	Optional (Same as PF)
SR-IOV Extended Capability	Optional	Not permitted
PASID Extended Capability	Optional	Not permitted
ATS Extended Capability	Optional	Optional (Same as PF)
Page Request Extended Capability	Optional	Not permitted

A3.3.2.1 Type 0 Configuration Space

The Device ID and Vendor ID of the Physical Function should be sufficient for system software to identify a unique instantiation of an AMU with a specific set of AHAs, in order to bind a device specific driver.

The Revere-AMU Physical Function requests a 64-bit prefetchable BAR0. This BAR defines the address space, which contains the management registers and any software Accelerator Message Interfaces (AMI-SWs) mapped to the PF (see A3.3.3 *PF/VF BAR Space*). This BAR address space may optionally have additional IMPLEMENTATION DEFINED contents, which can be mapped at specific locations (see Table A3.11).

The Revere-AMU Physical Function can request additional 64-bit prefetchable BAR resources for the purpose of MSI-X structures. It may also request additional 64-bit BAR resources with IMPLEMENTATION DEFINED contents, which have no additional restriction and can be prefetchable or non-prefetchable.

A3.3.2.2 PCI Power Management Capability

All Revere-AMU Functions implementing the PCI Power Management Capability must comply to the requirements defined by the PCI Express Base Specification [2], as well as the requirements related to power management defined in A3.14 *Power Management*.

When any component of an on-chip Revere-AMU implementation, such as the AMU, its associated AHAs or any other part supports always-on power domain wake events as defined in SBSA [3]:

- The Physical Function must support PCI Express PME generation
- The wake events must be controlled by the Physical Function PCI Power Management Capability, which means the Function must generate a wake event according to PME_En and it must be reflected in PME_Status
- The Physical Function PCI Power Management Capability must have the corresponding bits of the Power Management Capabilities Register PME_Support field set to one

See A3.14 *Power Management*.

A3.3.2.3 MSI-X Capability

If the Revere-AMU device supports interrupts, all Revere-AMU Functions must implement the MSI-X capability (see A3.7.7 *Interrupts*).

A3.3.2.4 SR-IOV Extended Capability

The Revere-AMU Physical Function may implement the SR-IOV extended capability, as defined by the PCI Express Base Specification [2]. SR-IOV support enables mapping of device contexts to independent Virtual Machines.

The Device ID of the Virtual Function (in the SR-IOV extended capability) should be sufficient for system software to identify a unique instantiation of an AMU with a specific set of AHAs, in order to bind a device specific driver.

The SR-IOV extended capability requests a 64-bit prefetchable VF BAR0. This BAR defines the address space, which contains the management registers and any software Accelerator Message Interfaces (AMI-SWs) mapped to that VF (see A3.3.3 *PF/VF BAR Space*). This BAR address space may optionally have additional IMPLEMENTATION DEFINED contents, which can be mapped at specific locations (see Table A3.11).

The SR-IOV extended capability may request additional 64-bit prefetchable VF BAR resources for the purpose of MSI-X structures. It may also request additional 64-bit BAR resources with IMPLEMENTATION DEFINED contents, which have no additional restriction and can be prefetchable or non-prefetchable.

A3.3.2.5 PASID Extended Capability

The Revere-AMU Physical Function may implement the PASID extended capability. This feature enables sub-assignment of device contexts to multiple applications within the same Function.

An implementation that does not implement the SR-IOV extended capability may still implement the PASID extended capability.

A3.3.2.6 ATS Extended Capability

If the Revere-AMU device supports accessing non-pinned memory, all Revere-AMU Functions must implement the ATS extended capability (see A3.7.9 *Accessing unpinned memory (SMMU page faults)*).

A3.3.2.7 Page Request Extended Capability

If the Revere-AMU device supports accessing non-pinned memory, the Revere-AMU Physical Function must implement the Page Request extended capability (see A3.7.9 *Accessing unpinned memory (SMMU page faults)*).

A3.3.3 PF/VF BAR Space

A3.3.3.1 Register map

Each Revere-AMU Function contains registers relating to the management interface and to the AMI-SW mapped to that Function.

These registers are mapped in the memory space of the Function defined by its BAR0.

The following tables describe the memory mapping of all the registers in the Physical and Virtual Functions depending on the AMI Type implemented and for two different configurations (Type A and Type B).

A3.3.3.1.1 High level view

The first table describes the mapping of various 64KB pages in the memory space of the Function.

- One Management 64KB page
- One or more AMI-SW related 64KB pages
- Zero or more optional 64KB pages with IMPLEMENTATION DEFINED content.
- Zero, one or two optional 64KB pages with MSI-X structure.

The number of AMI-SW Contexts per Function supported by an implementation is discoverable (A4.2.1 *Probing Commands and Responses*) and is equal to AMI_PER_F.

Function BAR0 Offset	Contents

Table A3.11: Function BAR0 map

Function BAR0 Offset	Contents
+TBD	Management page
+TBD	AMI-SW page #0
+...	...
+TBD + TBD * M	AMI-SW page #M
+TBD + TBD * (M + 1)	K optional TBD KB pages with IMPLEMENTATION DEFINED contents
+TBD + TBD * (M + 1 + K)	L optional TBD KB pages with MSI-X structures

In Table A3.11, the following apply:

- $M = \lceil \frac{AMI\_PER\_F}{16} \rceil$
- $K \geq 0$
- $0 \leq L \leq 2$

Note: For VFs this corresponds to BAR0 Region N. Region N is the memory space within VF BAR0 associated with VF N. Its offset from BAR0 VF_N starting address is computed as described in the PCI Express Base Specification [2].

Functions may optionally support IMPLEMENTATION DEFINED contents in memory space, which can appear:

- In BAR0, after the last supported AMI-SW, at offset TBD KB + (TBD KB * (M+1)).
- Anywhere in other device specific BARs.

A3.3.3.1.2 Management registers

The following table shows where the architected management registers appear in the Physical and Virtual Functions depending on the AMI Type implemented (Type A / Type B):

Table A3.12: Management Registers Memory Map

F BAR0 offset	Contents	Name (Type A)	Name (Type B)
+TBD	Management Registers	AMU_IDR - R/O	AMU_IDR - R/O
+TBD		AMU_AIDR - R/O	AMU_AIDR - R/O
+TBD		AMU_IIDR - R/O	AMU_IIDR - R/O
+TBD		AMU_CR	AMU_CR
+TBD		AMU_SR	AMU_SR
+TBD		MGT_RX_VECTOR[1]	MGT_RX_VECTOR
+TBD		MGT_TX_VECTOR[2]	MGT_TX_VECTOR
+TBD		RES0	MGT_TYPEB_BASE_PTR

[1] Only if the Revere-AMU implementation supports interrupts.
[2] Only if the Revere-AMU implementation supports interrupts.

F BAR0 offset	Contents	Name (Type A)	Name (Type B)
+TBD		MGT_RX_IRQ_CTRL[3]	MGT_RX_IRQ_CTRL
+TBD		MGT_TX_IRQ_CTRL[4]	MGT_TX_IRQ_CTRL
+TBD		MGT_TRACE_RX_CTRL[5]	MGT_TRACE_RX_CTRL
+TBD		ERR_IRQ_CTRL[6]	ERR_IRQ_CTRL[7]
+TBD	Command Management Socket Register	MSK_BASE_PTR	MSK_BASE_PTR
+TBD		MSK_CTRL	MSK_CTRL
+TBD	Response Management Socket Registers	MSK_BASE_PTR	MSK_BASE_PTR
+TBD		MSK_CTRL	MSK_CTRL
+TBD	Exception Management Socket Registers	MSK_BASE_PTR	MSK_BASE_PTR
+TBD		MSK_CTRL	MSK_CTRL
+TBD	Trace Management Socket Registers[8]	MSK_BASE_PTR	MSK_BASE_PTR
+TBD		MSK_CTRL	MSK_CTRL
+...	...	Reserved, RES0	Reserved, RES0
+TBD	Management AMI registers	TX_DIGEST	Reserved, RES0
+TBD		TX_DIGEST_MASK[9]	Reserved, RES0
+...	...	RES0	Reserved, RES0
+TBD		READ_INDEX_TX0	TX_DB_0
+...	...		Reserved, RES0
+TBD		WRITE_INDEX_TX0	Reserved, RES0
+...	...		Reserved, RES0
+TBD		RX_DIGEST	Reserved, RES0
+TBD		RX_DIGEST_MASK[10]	Reserved, RES0
+...	...	Reserved, RES0	Reserved, RES0

[3] Only if the Revere-AMU implementation supports interrupts.
[4] Only if the Revere-AMU implementation supports interrupts.
[5] Only if the Revere-AMU implementation supports tracing.
[6] Only if the Revere-AMU implementation supports interrupts.
[7] Only if the Revere-AMU implementation supports interrupts.
[8] Only if the Revere-AMU implementation supports tracing.
[9] Only if the Revere-AMU implementation supports interrupts.
[10] Only if the Revere-AMU implementation supports interrupts.

F BAR0 offset	Contents	Name (Type A)	Name (Type B)
+TBD		READ_INDEX_RX0	RX_DB_0
+TBD		READ_INDEX_RX1	RX_DB_1
+TBD		READ_INDEX_RX2[11]	RX_DB_2
+...	...	Reserved, RES0	Reserved, RES0
+TBD		WRITE_INDEX_RX0	Reserved, RES0
+TBD		WRITE_INDEX_RX1	Reserved, RES0
+TBD		WRITE_INDEX_RX2[12]	Reserved, RES0
+...	...	Reserved, RES0	Reserved, RES0
+TBD		Reserved, RES0	Reserved, RES0

For a detailed definition of each of the management registers refer to A3.4.3 *Management Registers*.

Reserved locations are RES0 as defined in [1].

A3.3.3.1.3 AMI-SW registers memory map

Each AMI-SW 64KB page contains the registers of 16 AMI-SWs mapped to that Function.

Table A3.13 describes the offset of the registers of AMI-SW #i (with i = 0..15) within one AMI-SW 64KB page, for the case where AMI-SW #i is mapped in the Function.

The number of sockets implemented is discoverable (by accessing the parameters NUM_TX_AMS and NUM_RX_AMS) and can be smaller than the maximum, 64. In that case, the registers that correspond to the sockets NUM_TX_AMS..63 and NUM_RX_AMS..63 shall be reserved, RES0.

Table A3.13: Map of the registers of the AMI-SW

Offset	Name (Type A)	Name (Type B)
+TBD + i * TBD	TX_DIGEST	Reserved, RES0
+TBD + i * TBD	TX_DIGEST_MASK[13]	Reserved, RES0
+...	Reserved, RES0	Reserved, RES0
+TBD + i * TBD	READ_INDEX_TX0	RX_DB_0
+TBD + i * TBD	READ_INDEX_TX1	RX_DB_1
+...	...	...
+TBD + i * TBD	READ_INDEX_TX63	RX_DB_63

[11] Only if the Revere-AMU implementation supports tracing.
[12] Only if the Revere-AMU implementation supports tracing.
[13] Only if the Revere-AMU implementation supports interrupts.

Offset	Name (Type A)	Name (Type B)
+TBD + i * TBD	WRITE_INDEX_TX0	Reserved, RES0
+TBD + i * TBD	WRITE_INDEX_TX1	Reserved, RES0
+...	...	...
+TBD + i * TBD	WRITE_INDEX_TX63	Reserved, RES0
+TBD + i * TBD	RX_DIGEST	Reserved, RES0
+TBD + i * TBD	RX_DIGEST_MASK[14]	Reserved, RES0
+...	Reserved, RES0	Reserved, RES0
+TBD + i * TBD	READ_INDEX_RX0	RX_DB_0
+TBD + i * TBD	READ_INDEX_RX1	RX_DB_1
+...	...	...
+TBD + i * TBD	READ_INDEX_RX63	RX_DB_63
+TBD + i * TBD	WRITE_INDEX_RX0	Reserved, RES0
+TBD + i * TBD	WRITE_INDEX_RX1	Reserved, RES0
+...	...	...
+0x0FF8 + i * 0x1000	WRITE_INDEX_RX63	Reserved, RES0

Reserved locations are RES0 as defined in [1].

When AMI-SW #i is not mapped in the Function, accesses to the corresponding 4KB page locations behave in the following way:

- Reads return value 0x41 for all data bytes
- Writes are ignored (WI as defined in [1])

Note

A Function driver can rely on all bytes of a 4KB page being equal to 0x41 to detect AMI-SW, which are not mapped to the Function.

[14] Only if the Revere-AMU implementation supports interrupts.

A3.4 Management Interface

A3.4.1 Overview

The management interface is used by Revere-AMU Function drivers to configure the device. This interface comprises:

- A set of memory-mapped registers.
- A special purpose AMI-SW, called management AMI, and an associated set of management messages exchanged through the management AMI.

The management interface is identical for the PF and VFs. VF drivers, however:

- Cannot modify certain bits in the AMU Control Register (AMU_CR).
- Have access to only a subset of the management messages that are exchanged through the management AMI.

These restrictions ensure that VF drivers can only manage the configuration related to the VF that they control.

A3.4.2 Management AMI

The management interface contains a management AMI, in addition to the management registers.

The management AMI is always enabled.

Unlike AMI-SW, the management AMI is not configured by messages but with the management registers (see A3.4.3 *Management Registers*).

The management AMI can be of type A1, A2 or B, and always matches the type implemented by the AMU for AMI-SWs (see A3.7.5 *AMI-SW types*). The AMI type can be discovered through the AMU_IDR register.

For type A1 and A2, the management AMI contains registers and AMS structures, and so only for the management sockets. Locations for the other, unused sockets are reserved, RES0.

For type B, the management AMI contains registers and doorbells, and the MGT_TYPEB_BASE_PTR register points to the Management TYPEB_AMI_SW table in memory. The Management TYPEB_AMI_SW table is similar to a TYPEB_AMI_SW table with a single entry, which contains AMS structures only for the management sockets (see A3.7.5.3 *AMI-SW of type B*). Memory locations for the other, unused sockets are reserved, RES0.

Apart from those specificities, the management AMI has the same layout as an AMI-SW (see A3.7.5 *AMI-SW types*).

A3.4.2.1 Management Sockets

There are three or four sockets in the management AMI:

Table A3.14: Management sockets of the management AMI

Socket	Direction	AMS
Command	Tx	TX AMS 0
Response	Rx	RX AMS 0
Exception	Rx	RX AMS 1
Trace[15]	Rx	RX AMS 2

[15] Only if the Revere-AMU implementation supports tracing.

Command messages are sent by Function drivers through the Command AMS. The set of commands available to the PF driver is a superset of those available to VF drivers. Commands are always acknowledged by the AMU through response messages. Function drivers must ensure that there is enough room in the response AMS for every outstanding command. When a command message is sent by software, the AMU must ensure that the update to the read index is performed after it has been processed but before the response is sent to the response AMS.

Exceptions are delivered by the AMU to the exception AMS in a Function driver when a given condition occurs. For example, an exception is delivered when profiling is configured for an ASN and a performance counter overflows.

Traces are delivered by the AMU to the trace AMS when a message is sent through an ASN that has been configured to be traced. Tracing is an optional feature of the architecture (see A3.9 *Tracing*). When an implementation does not support tracing, the trace AMS must not be implemented and software must ignore it.

Sockets of the management AMIs are always enabled. Management AMIs can be quiesced as part of Function quiescing (see A3.4.4.4 *Enable and disable operations*).

The AMU performs memory accesses on the ring slot array or TYPEB_AMI_SW table (when the AMI type implemented is type B) associated with the management AMI when a message is sent or received from one of the sockets in the management AMI. Accesses to these addresses are translated by an SMMU according to the translation scheme assigned to the corresponding Function, with no PASID.

The ring buffers for the Response and Exception management Sockets have an operating mode, which is fixed to back-pressure (see A3.7.2 *Ring buffers*). When the Revere-AMU implementation supports tracing, the operating mode of the Trace management Socket ring buffer can be configured with a dedicated MGT_TRACE_RX_CTRL register.

Management Sockets each have a set of management Socket registers to configure their parameters (see A3.4.3.13 *Management Socket Registers*).

The AMU triggers a Function error in any of the following cases:

- The Response AMS is full at the point where the AMU attempts to respond to a valid command. Clearing AMU_SR will cause the AMU to re-attempt the delivery of the response message.
- The Exception AMS is full at the point where the AMU attempts to deliver an exception message. Clearing AMU_SR will cause the AMU to re-attempt the delivery of the exception message.
- The AMU faults when accessing any of the base pointers associated with each of the management AMS.

Refer to Table A3.45 for details on DMA accesses.

A3.4.3 Management Registers

The management interface contains a set of memory mapped registers that are used in conjunction with the management AMI by Function drivers to configure the AMU (see A3.4.4 *Management Messages*).

Management registers appear at offset 0 within BAR0 for the Physical Function or at offset 0 within the region allocated in VF BAR0 for a particular Virtual Function (see A3.3.3 *PF/VF BAR Space*).

These registers can be directly accessed by the Function driver, and the controls they expose include the ability for Function drivers to:

- Discover a small set of architecture and implementation defined capabilities required for bootstrapping purposes.
- Be notified of errors.
- Configure the management AMI (see A3.4.4 *Management Messages*).
- When implemented, enable/disable tracing and profiling for the local Function.

In addition to the above, the PF uses management registers to:

- Enable/disable the AMU.
- When implemented, enable/disable tracing globally.

While the layout of the management registers is the same for the PF and VFs (A3.3.3 *PF/VF BAR Space*), certain bits in the AMU Control Register AMU_CR can only be accessed by the PF and behave as RES0 for VFs.

The following chapters describe the content of all management registers.

A3.4.3.1 AMU_IDR

Read-only

Bit	Name	Meaning
[63:32]	Reserved	RES0 as defined in [1]
[31:28]	MIN_LOG2_MSG_LENGTH	Minimum size of messages (in Doublewords as $\log_2$(MSG_LENGTH)) supported by the implementation Minimum value is 3, meaning 8 Doublewords
[27:24]	MAX_LOG2_MSG_LENGTH	Maximum size of messages (in Doublewords as $\log_2$(MSG_LENGTH)) supported by the implementation
[23:19]	MAX_LOG2_SIZE	Maximum size of the ring in slots as $\log_2$(SIZE) supported by the implementation
[18]	ASN_PROF	ASN profiling is supported by this implementation
[17]	TRACING	Tracing is supported by this implementation
[16:12]	AMI_TYPE	AMI type implemented 0b00000: Type A1 0b00001: Type A2 0b00010: Type B All others values are reserved
[11:6]	NUM_RX_AMS_M1	Number of RX sockets (AMSs) per interface (AMI) minus one NUM_RX_AMS = NUM_RX_AMS_M1 + 1
[5:0]	NUM_TX_AMS_M1	Number of TX sockets (AMSs) per interface (AMI) minus one NUM_TX_AMS = NUM_TX_AMS_M1 + 1

This register identifies architected capabilities of this Revere-AMU implementation.

A3.4.3.2 AMU_AIDR

Read-only

Bit	Name	Meaning
[63:8]	Reserved	RES0
[7:4]	ArchMajorRev	
[3:0]	ArchMinorRev	

This register identifies the Revere-AMU architecture version to which this implementation conforms.

A3.4.3.3 AMU_IIDR

Read-only

This register provides information about the implementation and implementer of the AMU.

A3.4.3.4 AMU_CR

Bit	Name	Meaning
[63:19]	Reserved	RES0
[18]	ASN_PROF_EN	Enable ASN performance profiling for this Function
[17]	TRACE_EN	Enable AMU generation and delivery of traces to the Function driver
[16:2]	Reserved	RES0
[1]	VF_TRACE_EN	Enable AMU generation and delivery of traces to VF drivers. This bit is RES0 for VFs.
[0]	AMU_EN	Enable AMU operations. This bit is RES0 for VFs.

This register manages the high-level configuration of the AMU. AMU_EN and VF_TRACE_EN are R/W for the PF and RES0 for VFs.

When the Revere-AMU implementation does not support profiling, ASN_PROF_EN is reserved, RES0. When the Revere-AMU implementation does not support tracing, TRACE_EN and VF_TRACE_EN are reserved, RES0.

A3.4.3.5 AMU_SR

This register is used for the notification of global and Function errors and other information related to the state of the Function (see A3.11 *Error detection, reporting and handling*).

Bit	Name	Meaning
[63:16]	Reserved	RES0
[15:0]	ERROR_STATE	Error state 0x0000: No error. 0x0001: Fault when accessing Command AMS. 0x0002: Fault when accessing Response AMS. 0x0003: Fault when accessing Exception AMS. 0x0004: Fault when accessing Trace AMS. 0x0005: Response AMS is full. 0x0006: Exception AMS is full. 0x0007: Invalid command message sent to the AMU through the Command AMS. 0x0008: Fault when accessing MGT_TYPEB_BASE. 0x0009: Fault when accessing TYPEB_AMI_SW. 0x000A: Fault when accessing PROF_TBL_BASE. 0x000B-0x0FFF: Reserved. 0x1000-0xFFFF: IMPLEMENTATION DEFINED

When a global or Function error is triggered, ERROR_STATE is updated by the AMU and the affected Function, or the whole device in the case of global errors, becomes inactive until the condition is corrected.

Writing a 0 to ERROR_STATE clears the current error condition.

When the Revere-AMU implementation does support interrupts, a change of value of AMU_SR for a non-zero value can trigger an interrupt, if configured in the ERR_IRQ_CTRL register.

Reset value: 0.

Refer to Table A3.45 for details on DMA accesses.

A3.4.3.6 MGT_RX_VECTOR

Bit	Name	Meaning
[63:31]	Reserved	RES0
[30:0]	RX_VECTOR	Interrupt vector number for management AMI Rx AMSs

This register allows to program the number of the vector to use when generating an interrupt related to RX_AMSs for the management AMI. See A3.7.7 *Interrupts* and A3.4 *Management Interface*.

When the Revere-AMU implementation does not support interrupts, this register is reserved, RES0. Otherwise, its reset value is IMPLEMENTATION DEFINED.

A3.4.3.7 MGT_TX_VECTOR

Bit	Name	Meaning
[63:11]	Reserved	RES0
[10:0]	TX_VECTOR	Interrupt vector number for management AMI Tx AMSs

This register allows to program the number of the vector to use when generating an interrupt related to TX_AMSs for the management AMI. See A3.7.7 *Interrupts* and A3.4 *Management Interface*.

When the Revere-AMU implementation does not support interrupts, this register is reserved, RES0. Otherwise, its reset value is IMPLEMENTATION DEFINED.

A3.4.3.8 MGT_TYPEB_BASE_PTR

Bit	Name	Meaning
[63:12]	MGT_TYPEB_BASE	Bits [63:12] of the address of the Management AMI TypeB table
[11:0]	Reserved	RES0

This register contains the 64-bit address of the Management TYPEB_AMI_SW table in memory, which contains the digests and the AMS data structures for the management AMI when it is of type B.

See A3.7.5.3 *AMI-SW of type B* and A3.4 *Management Interface*.

Accesses to this address are translated by an SMMU according to the translation scheme assigned to the

corresponding Function.

On a fault when the AMU accesses this pointer, a Function error is triggered (see A3.11.2.2 *Run-time errors*).

When the AMI type implemented by the AMU is A1 or A2, this register is reserved, RES0. Otherwise, its reset value is IMPLEMENTATION DEFINED.

Refer to Table A3.45 for details on DMA accesses.

A3.4.3.9 MGT_RX_IRQ_CTRL

Bit	Name	Meaning
[63:1]	Reserved	RES0
[0]	Enable	Controls interrupt generation for the Rx management Sockets 0: interrupt disabled 1: interrupt enabled (and is also subject to RX_DIGEST_MASK)

This register provides control over the generation of interrupts related to Rx management Sockets. The Enable bit of the MGT_RX_IRQ_CTRL has the same purpose for the management AMI than the RX_IRQ_ENABLE parameter of the AMI-SW configured by the PF-AMI-SW-CONFIGURE command. Changing the value of this register can generate an interrupt under circumstances described in A3.7.7 *Interrupts*.

When the Revere-AMU implementation does not support interrupts, this register is reserved, RES0. Otherwise its reset value is 0 (RX interrupt masked).

A3.4.3.10 MGT_TX_IRQ_CTRL

Bit	Name	Meaning
[63:1]	Reserved	RES0
[0]	Enable	Controls interrupt generation for the Tx management Sockets 0: interrupt disabled 1: interrupt enabled (and is also subject to TX_DIGEST_MASK)

This register provides control over the generation of interrupts related to Tx management Sockets. The Enable bit of the MGT_TX_IRQ_CTRL has the same purpose for the management AMI than the TX_IRQ_ENABLE parameter of the AMI-SW configured by the PF-AMI-SW-CONFIGURE command. Changing the value of this register can generate an interrupt under circumstances described in A3.7.7 *Interrupts*.

When the Revere-AMU implementation does not support interrupts, this register is reserved, RES0. Otherwise its reset value is 0 (TX interrupt masked).

A3.4.3.11 MGT_TRACE_RX_CTRL

Bit	Name	Meaning
[63:1]	Reserved	RES0
[0]	RX_MODE	Controls Trace management Socket ring operating mode 0: Back-pressure 1: Overwriting

This register provides control over the operating mode of the Trace management Socket ring buffer (see A3.7.2 *Ring buffers*).

When the Revere-AMU implementation does not support tracing, this register is reserved, RES0. Otherwise, its reset value is 0 (back-pressure operating mode).

A3.4.3.12 ERR_IRQ_CTRL

Bit	Name	Meaning
[63]	Enable	Controls interrupt generation in case of error 0: interrupt disabled 1: interrupt enabled
[62:31]	Reserved	RES0
[30:0]	Vector	Interrupt vector number to generate in case of error

This register provides control over the generation of interrupts related to errors, when the value of the AMU_SR register changes for a non-zero value. Changing the value of the ERR_IRQ_CTRL register can generate an interrupt, when the AMU_SR register has a non zero value.

When the Revere-AMU implementation does not support interrupts, this register is reserved, RES0. Otherwise its reset value is 0.

A3.4.3.13 Management Socket Registers

Management Sockets each have a set of configuration registers.

Table A3.26: Management Socket Registers.

Offset	Register
+TBD	MSK_BASE_PTR
+TBD	MSK_CTRL

Reserved locations are RES0 as defined in [1].

A3.4.3.13.1 MSK_BASE_PTR

This register contains the 64 bit address of the memory, which contains the ring buffer slot array for the corresponding management Socket. Accesses to this address are translated by an SMMU according to the translation scheme assigned to the corresponding Function.

Reset value: IMPLEMENTATION DEFINED.

Refer to Table A3.45 for details on DMA accesses.

A3.4.3.13.2 MSK_CTRL

Bit	Name	Meaning
[63:13]	Reserved	RES0

Bit	Name	Meaning
[12:9]	THRESHOLD	Controls the change of TX_DIGEST or RX_DIGEST bits Reset value is 0
[8:5]	LOG2_MSG_LENGTH	Maximum size of the management messages supported by the implementation in Doublewords as $\log_2$(MSG_LENGTH) This field is Read-only
[4:0]	LOG2_SIZE	Size of the ring in slots as $\log_2$(SIZE) Reset value is 0

This register allows to configure the Management Interface Socket parameters. See A3.7.3.1 *Threshold*) and A3.7.2 *Ring buffers*.

The maximum value for LOG2_MSG_LENGTH is 9 (e.g. the maximum size of Revere-AMU management messages is 2^9 Doublewords or 4KB).

LOG2_MSG_LENGTH should be dimensioned to hold any of the management messages, including IMPLEMENTATION DEFINED messages.

A3.4.4 Management Messages

The management interface contains a management AMI (configured through the management registers, see A3.4.3 *Management Registers*) and an associated set of architected messages.

The management interface specifies a set of architected messages used for:

- Probing the capabilities of the device.
- PF driver-VF driver communication.
- Virtual Function Management.
- AMI Management.
- ASN Management.
- AMS Management.
- Exception notifications.
- Tracing.

Management messages are divided into command, response, exception and trace messages. Each message type is associated with an AMS in the management AMI. Commands are sent by Function drivers to manage the configuration of a Function, AMI, ASN or AMS, or to retrieve the architected capabilities of the device. Commands always have an associated response, sent by the AMU. Exceptions are sent asynchronously by the AMU when a given condition takes place. Traces are delivered by the AMU when the architecture supports tracing and a given ASN is configured to be traced.

Certain command messages are reserved for the PF driver to use. In particular, those that:

- Configure the mapping of AMI-SW and AMI-HW to Functions.
- Set up and tear down sessions (ASNs) between AMIs.
- Configure and modify certain properties of sessions including stashing and QoS controls.

This section describes the management interface and the configuration that the AMU is expected to maintain. For a detailed list of commands and responses and associated formats, arguments and response values see Chapter A4 *Management Commands and Responses*. For a detailed list of exceptions and associated formats see Chapter A5 *Exceptions*.

A3.4.4.1 Revere-AMU Internal State

The management AMI is used to exchange messages with the AMU that modify its internal configuration, also referred to as state. The following diagram shows an overview of a Revere-AMU device and the state that the AMU must maintain internally.

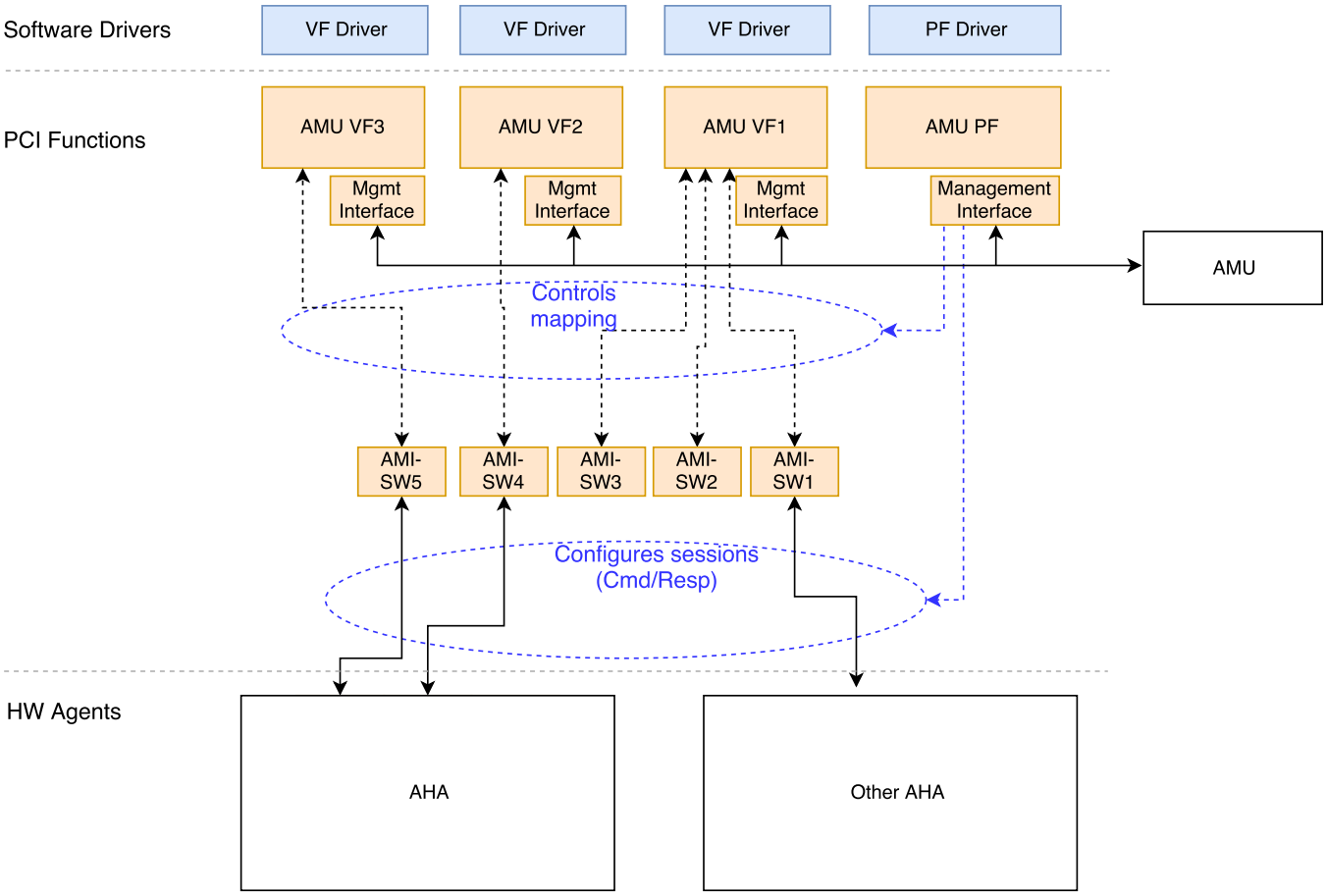

Figure A3.8: High level overview of a Revere-AMU device. The AMU maintains state related to AMIs, ASNs and AMSs.

While the internal state fundamental to the operations of the AMU is architected, it is IMPLEMENTATION DEFINED how such state is physically managed. An implementation may also extend the management interface to include messages that manage additional IMPLEMENTATION DEFINED state.

Note

Internal state refers to state entirely maintained by the AMU that cannot be directly observed by software. This includes all state except that which is maintained in BAR registers or in in-memory tables under the control of software, such as the TYPEB_AMI_SW table in certain implementations.

For example, a lightweight Revere-AMU implementation could manage all the state in internal registers or SRAM. A more sophisticated implementation could rely on normal memory and a set of IMPLEMENTATION DEFINED data structures instead, requiring additional messages between Function drivers and the AMU to set their associated base pointers and other configuration parameters.

Accelerator Message Interfaces for software and hardware (AMI-SWs and AMI-HWs) are groupings of AMU Sockets (AMSs) with independent interrupt and page-table based controls to enable sub assignment to user space software. Each AMI has zero or more sockets (AMS) for receiving and transmitting messages. An AMU Session (ASN) connects two AMSs in the system.

AMIs are mapped to Functions (see A3.4.4.3 *Mapping AMIs to Functions*). When a given AMI is mapped to a Virtual Function, the VF driver manages the configuration associated with that AMI, as well as the configuration associated with the AMSs contained in the AMI.

The PF driver can manage the global configuration of the AMU, including:

- AMU Sessions (ASNs).
- Accelerator Message Interfaces (AMIs), owned by the PF itself or by any arbitrary Function in the system.
- AMU Sockets (AMSs), owned by the PF itself or by any arbitrary Function in the system.

Management commands are used by Function drivers to probe and configure the AMU. The Revere-AMU configuration can be divided into:

- AMI-SW configuration.
- AMI-HW configuration.
- ASN configuration.
- AMS configuration.

The Revere-AMU configuration can be modified by the owning Function driver or reserved for the PF driver. The following sections describe the various state and captures the subset that is configurable by the owning Function driver.

A3.4.4.1.1 Constants fixed by implementation

The management interface provides a set of commands for Functions to probe the architected capabilities of the device that are not discoverable through the management registers (see A3.4.3 *Management Registers*).

The following table captures these capabilities:

Name	VF Driver Visible	Description
MAX_AMI_SW		Number of physical AMI-SW supported
MAX_ASN		Maximum number of ASN supported
NUM_AHA		Number of AHA associated with the AMU
PASID_SUPPORTED	X	PASID is supported by the implementation[16]
PASID_WIDTH	X	PASID width supported by the implementation[17]
ASN_PROF_N_COUNTERS	X	Number of ASN counters implemented (see A3.10 *Profiling*)
AMI_PER_F	X	Maximum number of AMI supported per Function

The PF-PROBE, PF-PROBE-AHA and F-PROBE commands can be sent by the PF and VF drivers to probe the PF-visible and VF-visible capabilities respectively. Refer to A4.2.1 *Probing Commands and Responses* for a detailed list of commands and responses.

A3.4.4.1.2 AMI-SW Configuration

The AMI-SW configuration is shared between the PF and VF drivers. The PF driver performs AMI-SW management either statically or as a result of IMPLEMENTATION DEFINED requests from a VF.

An AMI-SW captures the state that:

- Determines the mapping of that AMI-SW to a Function, including the ID to be used within that Function.

[16] mirrors the PCI Express PASID capabilities [2].
[17] mirrors the PCI Express PASID capabilities [2].

- Controls the PASID to be used for the translation scheme, in addition to that used by the Function.
- Controls interrupt vectors.

Name	VF Driver Controllable	Description
F_OWNER		Function the AMI-SW is mapped to
AMI_SW_ID		Local ID within Function
ENABLE	X	AMI-SW is enabled/disabled
PASID_ENABLE	X	PASID is enabled/disabled for this AMI-SW
PASID	X	PASID Value
TX_VECTOR	X	Interrupt vector number for TX AMSs
RX_VECTOR	X	Interrupt vector number for RX AMSs

Refer to A4.2.3 *AMI-SW Management Commands and Responses* for a detailed list of commands.

Refer to Table A3.45 for details on DMA accesses.

A3.4.4.1.3 AMI-HW Configuration

The AMI-HW configuration is shared between the PF and VF drivers. The PF driver performs AMI-HW management either statically or as a result of IMPLEMENTATION DEFINED requests from a VF driver.

An AMI-HW captures the state that:

- Determines the mapping of that AMI-HW to a Function, including the ID to be used within that Function.
- Controls the translation scheme.
- Configures the credit size to be used for that AMI-HW (see Chapter B2 *AMU AHA Interface protocol*).

Name	VF Driver Controllable	Description
F_OWNER		Function the AMI-HW is mapped to
AMI_HW_ID		Local ID within Function
LOG2_HW_CRED_SIZE		Credit size (in $\log_2$ doublewords)to use for this AMI-HW
ENABLE	X	AMI-HW is enabled/disabled
PASID_ENABLE	X	PASID is enabled/disabled for this AMI-HW
PASID	X	PASID Value

Refer to A4.2.4 *AMI-HW Management Commands and Responses* for a detailed list of commands.

Refer to Table A3.45 for details on DMA accesses.

A3.4.4.1.4 ASN Configuration

The ASN configuration is owned by the PF driver. The PF driver performs ASN management either statically or as a result of IMPLEMENTATION DEFINED requests from a VF driver.

An ASN captures the state that:

- Links an AMS in one AMI to another AMS in another AMI.
- Is used to generate appropriate cache stashing hints for in-band data.
- Enables differentiation across ASNs (QoS).
- Assigns a Function owner to an ASN for the purposes of tracing and profiling.
- Configures tracing and profiling for the ASN.

Name	VF Driver Controllable	Description
MFO		Message format to use for this ASN

Name	VF Driver Controllable	Description
MF_OB_BUF_NUM		Number of references to out-of-band buffers
LOG2_MSG_LENGTH		Message length (in Doublewords)
SOURCE		Source AMS within AMI and Function
DESTINATION		Destination AMS within AMI and Function
STASH_CTL		Stashing controls to use for this ASN
PRIO		Priority to use for this ASN
SRC_AMI_TYPE		Type of the source context
DST_AMI_TYPE		Type of the destination context
MON_OWNER		Function to use for tracing and profiling
STASH_DEST		Physical destination for stashing
TRACE_CTL	X	Tracing controls to use for this ASN
PROF_CTL	X	Profiling controls to use for this ASN
PROF_MASK	X	Profiling mask
PROF_TBL_BASE	X	Profiling table base pointer

SRC_AMI_TYPE and DST_AMI_TYPE determine the type of contexts that participate in the ASN. An ASN can connect:

- A software context with another software context.
- A software context with a hardware context.
- A hardware context with a software context.
- A hardware context with another hardware context.

SOURCE and DESTINATION determine the TX AMS and the RX AMS that are connected together through the ASN. Both pieces of configuration include:

- A Function owner.
- An AMI within the Function owner.
- An AMS within the AMI.

Other than the tracing and profiling, configuration associated with an ASN can not be modified after the ASN has been created.

Refer to A4.2.5 *ASN Management Commands and Responses* for a detailed list of commands.

A3.4.4.1.5 AMS Configuration

The AMS configuration is owned by the Function that contains the AMS. A Function driver can perform AMS management without intervention from the PF driver. Where an AMI-SW is sub-assigned, user space software can request the Function driver to configure the AMS on its behalf. The mechanisms used for the communication between user space software and a separate Function driver are IMPLEMENTATION DEFINED.

An AMS captures the state that:

- Determines whether the AMS has been configured.
- Configures the AMS base pointer.
- Configures certain AMS parameters, such as ring slot array address and size.

Name	VF Driver Controllable	Description
SIZE	X	Number of slots in the slot array
THRESHOLD	X	Controls the change of DIGEST bits
RING_BASE_PTR	X	64-bit virtual address of the slot array
RX_MODE	X	Back-pressure or overwriting

AMS management messages also enable Functions to modify tracing and profiling configuration parameters of the ASN that is associated with the AMS.

A3.4.4.2 AMU management command trapping

The AMU has the capability to transparently trap and forward through the PF management AMI arbitrary commands sent by a VF driver, and forward the architected response through a VF management AMI. This is done through a management command trapping capability that can be controlled using the management command PF-F-TRAP-CONFIGURE.

When enabled in a Function, trapping occurs for any of the following cases:

- An AMI management command is sent by a Function driver to the AMU where the AMI_SW_ID is not currently mapped in the Function.
- A PF management command is sent through a VF management AMI to the AMU.
- The Command opcode is in the IMPLEMENTATION DEFINED range and is not implemented by the AMU.

The purposes of this capability are:

- To provide a path through the device for configuration commands between the VF and PF drivers, as described in A2.2 *Device and I/O Virtualization Model*.
- To enable transparent overprovisioning by more privileged software.
- To enable device driver compatibility by trapping commands with unsupported opcodes.

The meaning of the IMPLEMENTATION DEFINED commands exchanged between the PF and VFs must be agreed between PF and VF drivers for a particular device.

The PF-F-TRAP-CONFIGURE command is used to enable or disable trapping of commands via the AMU. PF driver should disable trapping if a suitable software path exists (for example, para-virtualization) for communication between the PF and VF drivers and no AMI overprovisioning is needed.

The E-PF-TRAPPED-CMD exception message is sent to the PF when a command has been trapped (provided trapping of commands via the AMU is enabled).

The PF-F-TRAP-RESP command is then used to convey the response to the VF. It encapsulates the response command the VF driver will receive. This response command must use the proper opcode (response opcode for architected commands or IMPLEMENTATION DEFINED opcode for IMPLEMENTATION DEFINED commands) along with status bits. The status bits will be used by the AMU to raise an error if the value is non-zero (in which case the response is sent in the exception session).

Note

Exchanging commands between the PF driver and a VF driver is not required to configure the AMU itself. Arm recommends that a paravirtualised path is used for PF-VF communication where one exists, and the hardware path is disabled.

The following sequence diagram shows the commands exchanged by the PF driver, a VF driver and the AMU when the VF driver sends a command that is trapped:

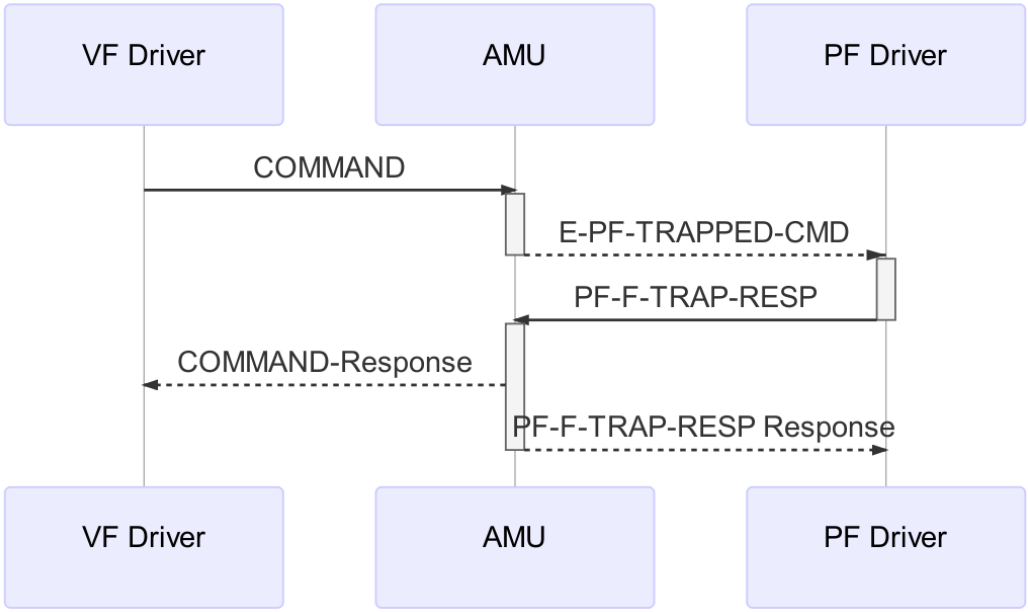

Figure A3.9: PF-VF Driver Trapping Sequence diagram.

Refer to A4.2.2 *Virtual Function Management Commands and Responses* for a detailed list of commands.

A3.4.4.3 Mapping AMIs to Functions

Software and hardware Accelerator Message Interfaces (AMI-SWs and AMI-HWs) are mapped to Functions by the PF driver through the F_OWNER and AMI ID fields that are part of the AMI management commands. When mapped, a physical AMI-SW or AMI-HW is owned by the Function and associated locally with the Function through an AMI-SW ID or AMI-HW ID. This ID can be used to retrieve or modify the configuration of the AMI as well as to reset, enable and disable the AMI.

When the AMI to be mapped is an AMI-SW, the AMI-SW ID also determines the offset of its associated registers in the BAR space. The offset is 4KB x AMI_SW_ID + 64KB (see A3.3.3 *PF/VF BAR Space*).

AMI-SW IDs are assigned according to the following rules:

- AMI-SW IDs must be assigned uniquely per Function.
- Sparse allocation of the AMI-SW ID is permitted.
- The highest AMI-SW ID assigned (at AMI_SW_ID * 4KB + 4KB), must be within the BAR0 or VF BAR0 aperture size.

Note

Sparse allocation of the AMI-SW IDs is useful to align independent AMI-SW to a 64KB page, so that they can be independently assigned to software using the PASID mechanism.

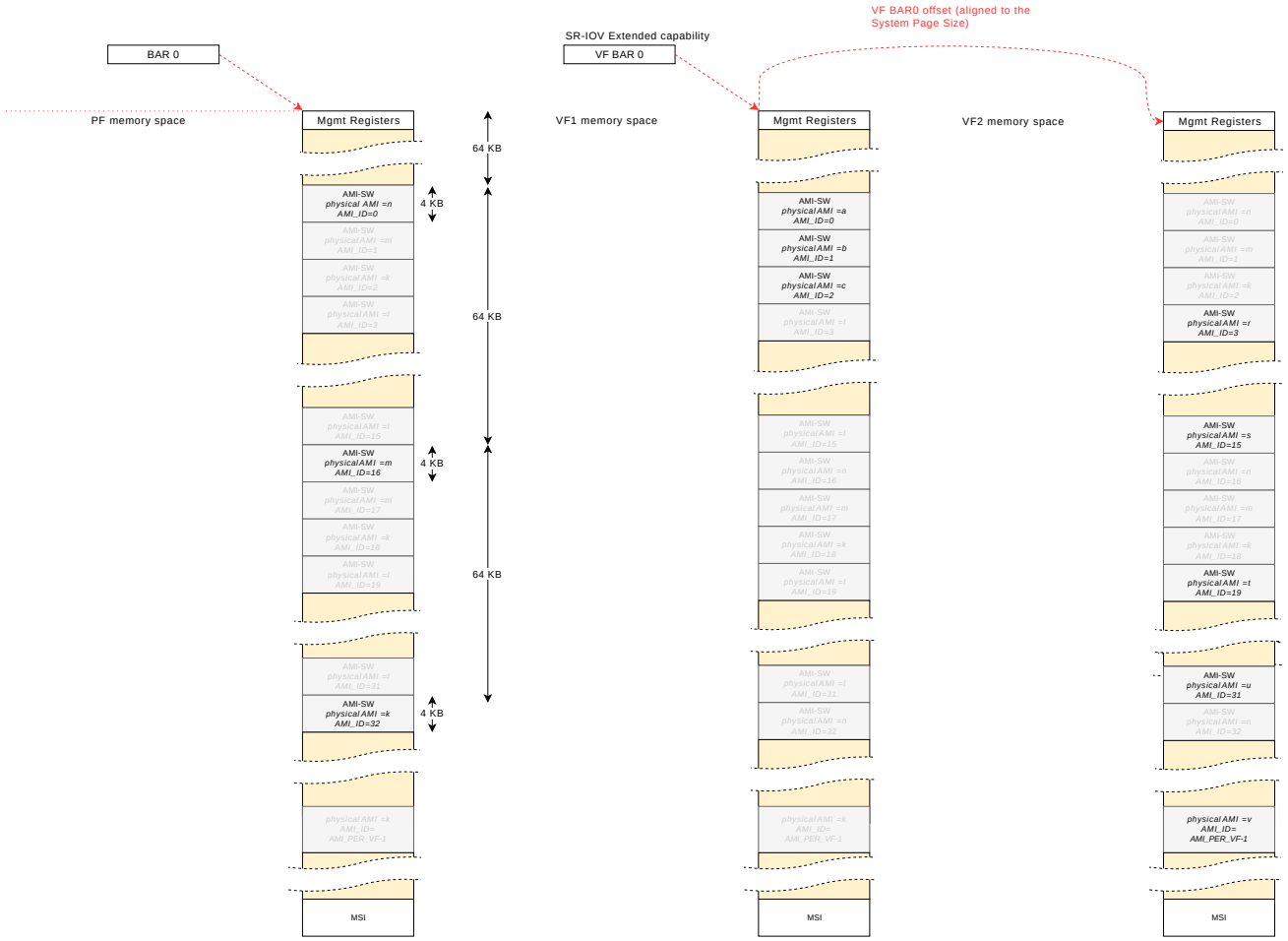

Figure A3.10: The location of AMI-SW registers in the system address space is determined by BAR0 (for the PF) and VF BAR0 in the SR-IOV extended capability of the Revere-AMU PF (for VFs) and the Function Owner and AMI-SW ID associated with a given AMI-SW.

VF BAR0 of the SR-IOV Extended capability points to the VF1 memory space. The offset between two VF memory spaces is determined by the VF BAR0 aperture size which shall be a multiple of the System Page Size as specified by the SR-IOV Extended capability.

Note

Hardware is permitted to implement a mapping scheme where not every physical AMI-SW can be mapped to a given AMI-SW ID in a given Function. This allows implementations to trade off flexibility of allocation and complexity, for example using a CAM, or a sparse mapping scheme for registers relating to an AMI-SW.

The PF driver is expected to have knowledge of implementation specific restrictions to mapping.

Function drivers may require to retrieve the set of AMI-SW IDs and AMI-HW IDs mapped to physical AMI-SWs and AMI-HWs at runtime. This information is encapsulated into what is referred to as the Function AMI map. The AMI map can be retrieved by sending an F-GET-AMI-MAP command through the command AMS.

The format of this map is as follows:

- It has a total size of 16KB.
- The first half of the map (8KB) encodes the state associated with each AMI-SW ID.

- The second half of the map (8KB) encodes the state associated with each AMI-HW ID.
- In both cases, one bit is used to convey the state of each AMI, where “1” means that the AMI ID has a valid mapping and “0” means that the AMI ID is not currently mapped.
- Byte at offset N within the AMI-SW section and AMI-HW section of the AMI map represents respectively the state of AMI-SW IDs and AMI-HW IDs from N * 8 (least significant bit) to N * 8 + 7 (most significant bit).

Where an implementation does not implement the full local AMI ID space, locations in the map associated with non-implemented AMIs must be filled with zeroes.

A3.4.4.4 Enable and disable operations

The management interface provides a set of commands for Function drivers to enable/disable an AMI. The PF driver can also enable/disable an individual Function in the system.

A3.4.4.4.1 Enabling and disabling a Function

The PF driver can enable and disable individual Functions in the system. These operations are allowed regardless of the state of the Function, as long as it exists. When a Function is disabled:

- All AMIs within the scope of the disabled Function are quiesced (see A3.4.4.4.3 *AMI Quiescing*). This includes AMI-SWs and AMI-HWs mapped to the Function, as well as the management AMI.
- The AMU stops processing any further messages within the scope of the disabled Function.
- Any sessions within the scope of the disabled Function are not affected. Re-enabling the Function would cause the AMU to continue processing messages for those sessions.

Note

Disabling or enabling a Function as described in this section does not have any effect associated with the PCI Express Function as presented to software. The PCI Express Function remains present in the system and the BAR mappings remain untouched. The scope of the effect of these operations is limited to the AMI-SWs and AMI-HWs mapped to the Function as well as its management AMI.

Refer to A4.2.2 *Virtual Function Management Commands and Responses* for a list of commands and responses.

A3.4.4.4.2 Enabling and disabling an AMI

A Function driver can enable and disable individual AMIs mapped to it. When an AMI is disabled:

- All AMSs within the scope of the disabled AMI are quiesced (see A3.4.4.4.3 *AMI Quiescing*).
- The AMU stops processing any further messages within the scope of the disabled AMI.
- Any sessions within the scope of the disabled AMI are not affected. Re-enabling the AMI would cause the AMU to resume message processing for those sessions.

Refer to A4.2.3 *AMI-SW Management Commands and Responses* and A4.2.4 *AMI-HW Management Commands and Responses* for a list of commands and responses.

A3.4.4.4.3 AMI Quiescing

When an AMI is quiesced:

- The AMU should issue all transactions relating to in-flight messages associated with the AMI and not issue transactions relating to further messages from the TX sockets (AMS).
- The AMU should flush all buffered messages to the RX sockets (AMS) in a timely manner, issuing all required transactions.

Note

End-end crediting ensures that there is always sufficient buffer space at the receiver for all messages buffered within the AMU to be accepted.

Note

Arm recommends Revere-AMU implementations provide a guarantee that quiescing operations finish in bounded time.

When an AMI is quiesced, the AMU does not send a response message until:

- The AMU will not issue any further transactions relating to any socket (AMS) in scope.
- All transactions that have been issued are globally observed.

Note

The mechanism to ensure global observation of transactions is IMPLEMENTATION DEFINED. For example, if the AMU is part of a PCI Express endpoint, a zero-length read may be required to establish that posted transactions are globally observed.

A3.4.4.5 Reset domains

This section describes the behaviour of a Revere-AMU device when a PCI Express architected Function Level Reset is performed. The management interface also provides a set of commands for Function drivers to reset an AMI.

A3.4.4.5.1 PF Function level reset (FLR)

Reset of the Revere-AMU PF is triggered by:

- A fundamental reset.
- A Function level reset of the PF.

A reset of the Revere-AMU PF:

- Resets all state that is architecturally required to be reset by the PCI Express Base Specification [2].
- Disables all VFs (as is required by the PCI Express Base Specification [2]).
- Resets all VFs as per the below procedure, with the exception that VF reset notification messages are not generated.
- Resets management registers in the PF to their default values (see A3.4.3 *Management Registers*).

A3.4.4.5.2 VF Function level reset (FLR)

Reset of a Revere-AMU VF is triggered by:

- A fundamental reset.
- A Function level reset of the PF.
- A Function level reset of the VF.

A reset of a Revere-AMU VF:

- Resets all state that is architecturally required to be reset by the PCI Express Base Specification [2].
- Resets all AMI_SW configured as within that VF, as per the below procedure, including AMSes belonging to those AMIs.

- Resets all AMI_HW configured as owned by that VF, as per the below procedure, including AMSes belonging to those AMIs.
- Resets management registers in the VF to their default values (see A3.4.3 *Management Registers*).
- Causes a VF reset notification message to be sent to the PF.

Note

The VF reset notification message enables a software driver using the PF to clear out any state that may be associated with the current user of that VF. The fact that all AMI_SW are disabled on reset means that a race is prevented where a new user sends/receives messages before any previous state has been cleared in the PF.

A3.4.4.5.3 AMI-SW reset

Reset of an AMI-SW is triggered by:

- A fundamental reset
- A Function level reset of the PF
- A Function level reset of the Function the AMI-SW is mapped to
- An AMI reset command received by the AMU

A reset of an AMI-SW:

- Causes all buffered messages for all the RX and TX AMSes in this AMI-SW to be immediately discarded.
- Clears internal configuration state associated with the AMI-SW and AMSes in those AMIs and resets its registers to their default values (see A3.4.4.1.2 *AMI-SW Configuration* and A3.7 *Accelerator Message Interface for Software*). This includes putting the AMI-SW into a disabled state.

A3.4.4.5.4 AMI-HW reset

Reset of an AMI-HW is triggered by:

- A fundamental reset
- A Function level reset of the PF
- A Function level reset of the Function the AMI-HW is mapped to
- An AMI reset command received by the AMU

A reset of an AMI-HW:

- Causes any buffered messages to be immediately discarded.
- Clears internal configuration state associated with the AMI-HW and AMSes in those AMIs (see A3.4.4.1.3 *AMI-HW Configuration*). This includes putting the AMI-HW into a disabled state.
- Resets all AHA specific context state relating to that AMI-HW.

A3.4.4.6 AMI and AMS state machines

A3.4.4.6.1 AMI State machine

Following state machine summarizes the different management messages, when they are valid and how they affect the AMI state.

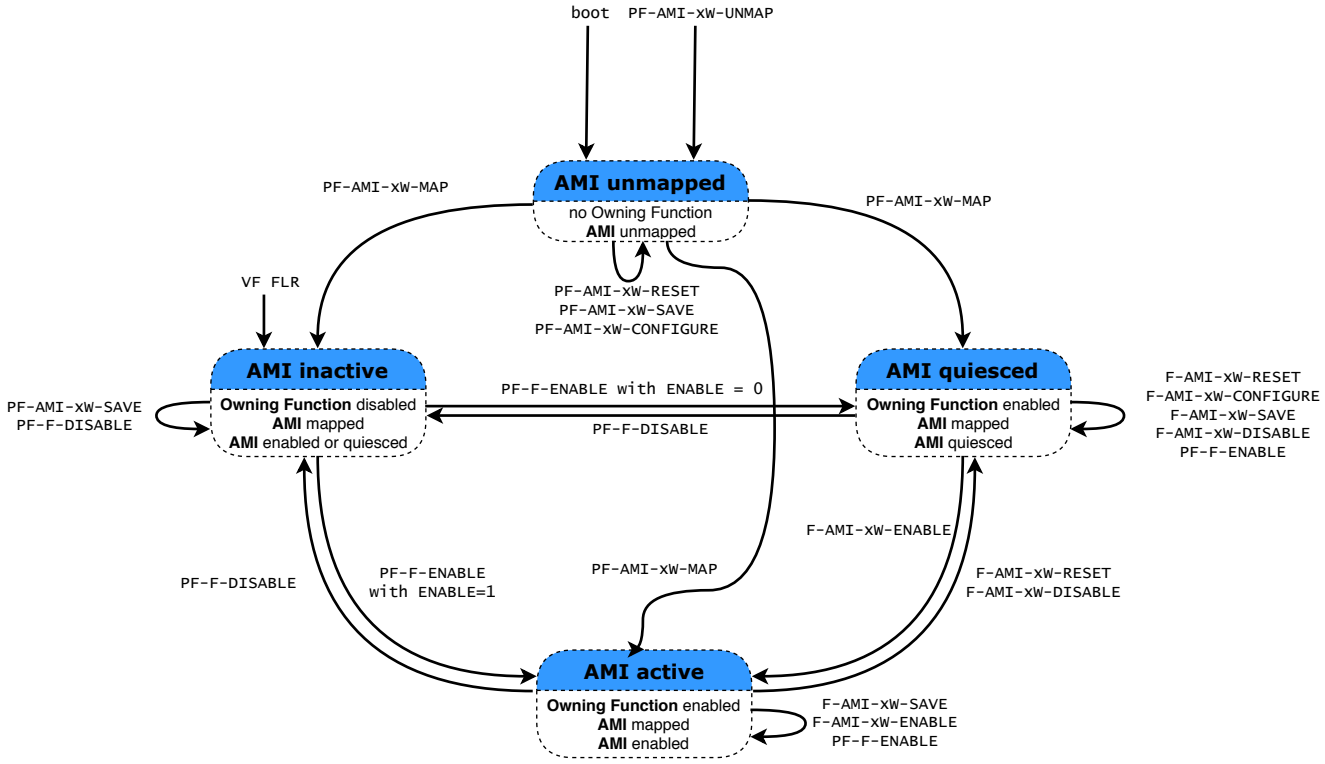

Figure A3.11: AMI State Machine.

A3.4.4.6.2 AMS State machine

Following state machine summarizes the different management messages, when they are valid and how they affect the AMS state.

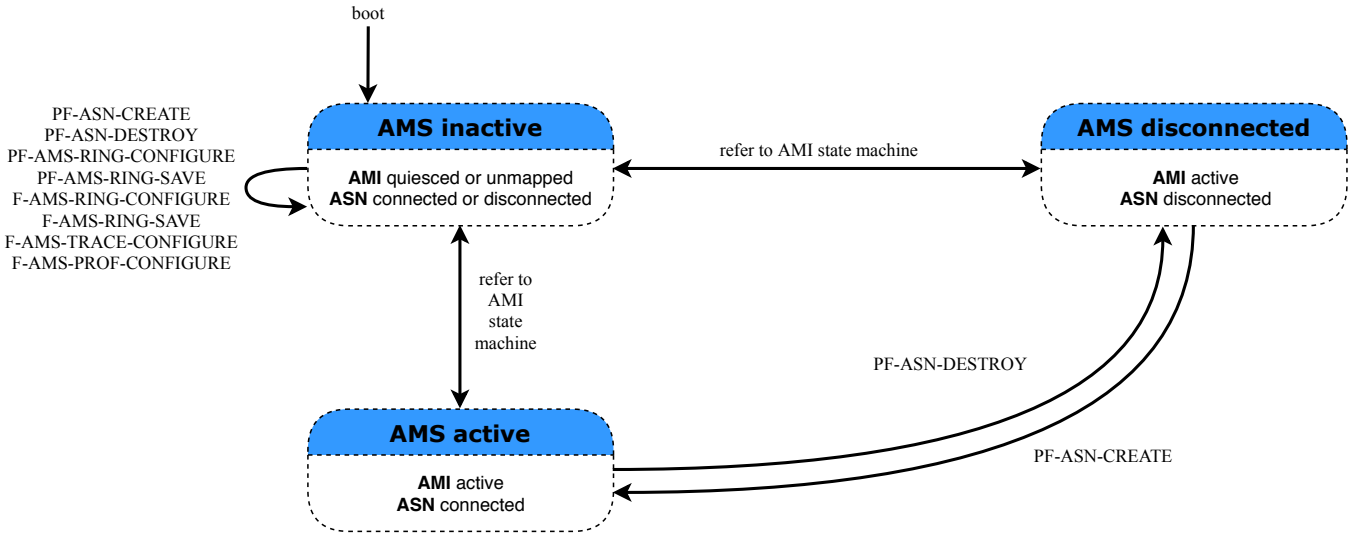

Figure A3.12: AMS State Machine.

A3.5 Cache stashing

Cache stashing is an optimization that allows a master to indicate that cache lines containing a particular datum, or data range should be placed in a particular cache in the system.

AMU architecture provides stashing controls to enable an implementation to generate stashing transactions as part of a message being sent, that cause specified components of the message to be allocated in a cache close to the message receiver.

Revere provides controls for stashing of:

- Revere message,
- Out-of-band buffer table, and
- Out-of-band buffers.

Stashing parameters are configured per session by the PF driver through the management interface (A4.2.5 *ASN Management Commands and Responses*) and are conveyed at message level for the out-of-band buffers.

- Stashing is enabled per session
- Stashing destination is controlled per session by the STASH_DEST parameter
- Stashing control for messages and the out-of-band buffer table is a property of the session
- Stashing control for out-of-band buffers is a property of the message descriptor, the out-of-band buffer table or the out-of-band buffer header depending on the message format in use

The description of which part of information should be stashed and where it should be placed is defined in the stash control information field.

The same stash control information field is used in both sessions (to control stashing of the messages and out-of-band buffer tables) and out-of-band buffer table (to control stashing of out-of-band buffers).

Table A3.33 gives an overview of which parameters control the data stashing, and to which data they apply:

Table A3.33: Cache Stashing Controls summary

Control	Message	Out-of-band buffer table	Out-of-band buffer
Session (ASN) STASH_ENABLE	X	X	X
Session (ASN) STASH_DEST	X	X	X
Session (ASN) STASH_CTL	X	X[18]	
Message descriptor OB_BUF_STASH_CTL			X (MFO2, MFO3)
Out-of-band buffer table OB_BUF_STASH_CTL			X (MFO3)
Out-of-band buffer header STASH_CTL			X (MFO4)

An invalid Stashing configuration must never cause a Revere-AMU device to issue invalid transactions.

A3.5.0.1 Stash Destination

The Cache Stashing Destination (STASH_DEST) field is a common 32-bit field used in both session configuration messages and in the optional pin-level interface for hardware accelerators.

[18] A Revere-AMU implementation is allowed to perform stashing of the out-of-band buffer table according to the session STASH_CTL, but it must ignore the STASH_CTL.STASH_LEN and STASH_CTL.STASH_OFFSET fields.

How a cache stashing destination is specified in a system is IMPLEMENTATION DEFINED, except for the cases described in the following sections.

A3.5.0.1.1 Stash Destination for AMBA 5 AXI

This section is OPTIONAL.

In the case of a system based on AMBA 5 AXI, the destination of cache stashing is expressed by the AWSTASH family of signals. Table A3.34 shows the mapping of the STASH_DEST field to the AMBA 5 AXI signals:

Table A3.34: Cache Stashing Destination mapping to AMBA 5 AXI signals

Bits	Signal	Description
[31:18]	- Reserved -	RES0
[17]	AWSTASHLPIDEN	Logical Processor Identifier is valid and should be used
[16:12]	AWSTASHLPID[4:0]	Logical Processor Identifier
[11]	AWSTASHNIDEN	Node Identifier is valid and should be used
[10:0]	AWSTASHNID[10:0]	Node Identifier

Refer to the AMBA AXI and ACE Protocol Specification [4] for valid signals combinations and details.

A3.5.0.1.2 Stash Destination for AMBA 5 CHI

This section is OPTIONAL.

In the case of a system based on AMBA 5 CHI, the destination of cache stashing is expressed by the Stash family of fields. Table A3.35 shows the mapping of the STASH_DEST field to the AMBA 5 CHI fields:

Table A3.35: Cache Stashing Destination mapping to AMBA 5 CHI fields

Bits	Field	Description
[31:18]	- Reserved -	RES0
[17]	StashLPIDValid	Logical Processor ID is valid
[16:12]	StashLPID	Logical Processor ID
[11]	StashNIDValid	Node ID is valid
[10:0]	StashNID	Node ID

Note

With AMBA 5 CHI setting both StashLPIDValid and StashNIDValid to zero is a valid combination, which allows to perform stashing without specifying a stash target.

In this case, the Home Node can stash to an RN, which has the data in its cache, effectively “following” the data

where it is in the system.

Refer to the ARM AMBA 5 CHI Architecture Specification [5] for valid fields combinations and details.

A3.5.0.2 Stash Control Information field

The Cache Stashing Control (STASH_CTL) field is a common field used in both session configuration messages and in the descriptor or out-of-band buffer table.

Table A3.36 includes a target cache, a stash request type and a description of two regions in the messages or out-of-band buffers.

Note

The possibility to define two separated regions should address most of the requirements encountered so far. One justification is the need to stash the head (header) and the tail (CRC) of a packet

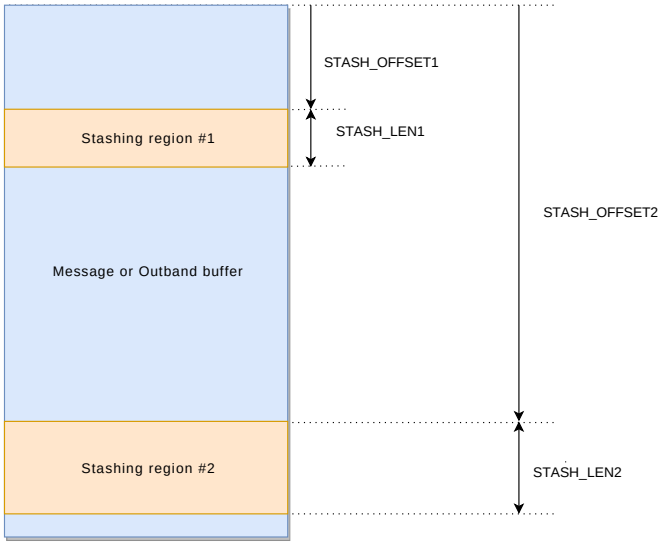

Figure A3.13: Control Stashing Regions

Table A3.36: Cache Stashing Control Information

Offset	Bits	Name	Description
+0x00	[31:19]	STASH_LEN1	Number of bytes to stash for the first stash region
+0x00	[18:6]	STASH_OFFSET1	Offset in bytes added to the start address (message or out-of-band buffer) to stash for 1st region
+0x00	[5:3]	STASH_TYPE	Type of stash request (up to 8 IMPLEMENTATION DEFINED possible values)
+0x00	[2:0]	- Reserved -	RES0
+0x04	[31:25]	- Reserved -	

Offset	Bits	Name	Description
+0x04	[24:12]	STASH_LEN2	Number of bytes to stash for the second stash region
+0x04	[11:0]	STASH_OFFSET2	Offset in bytes added to the start address (message or out-of-band buffer) to stash for 2nd region

A3.6 Translation stashing

To optimize system performance Revere-AMU and AHAs can implement a scheme to load a translation entry into the SMMU or any internal Address Translation Cache (ATC) before the address is actually used for a data transfer.

If the Revere-AMU system implements unpinned memory access, it can use information in Table A3.36 to prefetch translations and issue speculative Page Requests.

When PCI Express ATS (refer to A3.7.9 *Accessing unpinned memory (SMMU page faults)*) is not implemented or if the SMMU is tightly coupled, translation prefetching can be achieved using IMPLEMENTATION DEFINED mechanisms through DMA ports.

Translation stashing is one optimization for Revere-AMU translations. There can be others, such as SMMU prefetching (next or previous page). Those optimizations are IMPLEMENTATION DEFINED.

A3.7 Accelerator Message Interface for Software

A3.7.1 Overview

This section describes the interface to the device assignable context (AMI-SW) that enables software running on Arm v8 PE to send and receive messages.

Contexts executed on a PE exchange messages over established sessions (ASNs) through TX socket (TX AMS) for sending messages and RX socket (RX AMS) for receiving messages.

Sockets (AMSs) are implemented using ring buffers (described in A3.7.2 *Ring buffers*) which consist of a circular slot array, an associated set of configuration and status parameters and indices.

Software running on a PE:

- send messages by writing to the ring buffer of a TX socket (TX AMS).
- receive messages by reading from the ring buffer of a RX socket (RX AMS).

The number of TX socket (NUM_TX_AMS) and the number of RX socket (NUM_RX_AMS) per AMI-SW are IMPLEMENTATION DEFINED and discoverable by accessing the AMU_IDR in the memory-mapped management registers.

A3.7.2 Ring buffers

In Revere-AMU system architecture, a ring buffer is associated with exactly one socket (AMS).

Ring buffers consist of arrays of slots allocated in Normal memory and associated parameters.

Ring buffers are single-producer single-consumer.

The size of a ring buffer is determined as a number of slots restricted to a power of 2.

The size of the slot is configurable (in number of Doubleword) for each ring buffer and restricted to a power of 2.

Software enqueues (produces) messages to the ring buffer of a TX socket (TX AMS) and the AMU dequeues (consumes) messages from that ring buffer.

The AMU enqueues (produces) messages to the ring buffer of a RX socket (RX AMS) and the software dequeues (consumes) messages from that ring buffer.

A3.7.2.1 Ring Buffer Configuration

The ring buffer parameters are configured through the architected management messages PF-AMS-RING-CONFIGURE and F-AMS-RING-CONFIGURE:

- RING_BASE_PTR:
  - Virtual address pointing to the beginning of the slot array aligned on a Doubleword boundary. On a fault when the AMU accesses this pointer, the AMU triggers an E-F-AMI-RING-FAULT exception.
- RX_MODE:
  - Operating mode for a RX AMS (‘0’: BACK-PRESSURE, ‘1’: OVERWRITING). See A3.7.2.2.1 *Receive operating modes* for more details.
- THRESHOLD:
  - Set the threshold used to set and clear digest bits. See A3.7.3 *Digests* and A3.7.3.1 *Threshold*.
- LOG2_SIZE:
  - Describes the size of the ring in number of slots. Expressed as $\log_2$(SIZE).
  - Used to generate MASK = ((1U << LOG2_SIZE) - 1) that converts free-running 32-bit logical indices into physical slot indices in the slot array.
  - The maximum number of slots is determined by MAX_LOG2_SIZE in the AMU_AIDR.

– When LOG2_SIZE is out of range, the response to the ring configuration command (PF-AMS-RING-CONFIGURE or F-AMS-RING-CONFIGURE) returns an error.

Note

Limiting the possible slot sizes to a power of 2 might be limiting for some use-cases. This might be revisited based on feedback.

The size of the slot (LOG2_MSG_LENGTH) is a property of the session and is configured through ASN configuration messages A4.2.5 *ASN Management Commands and Responses*.

A Revere-AMU implementation supports a minimum and maximum message length sizes, MIN_LOG2_MSG_LENGTH and MAX_LOG2_MSG_LENGTH respectively, which are discoverable by software by accessing AMU_IDR.

The maximum value for MAX_LOG2_MSG_LENGTH is 9 (e.g. the maximum size of Revere-AMU messages is 2^9 Doublewords or 4KB).

The minimum value for MIN_LOG2_MSG_LENGTH is 8 Doublewords.

When LOG2_MSG_LENGTH is out of range, PF-ASN-CREATE returns an error code.

Refer to Table A3.45 for details on DMA accesses.

A3.7.2.2 Ring Buffer Operation

To operate ring buffers, the software and the AMU update read and write indices:

- READ_INDEX:
  - Unsigned 32-bit free-running logical slot index.
  - Indicates the position of the next element to read in the slot array.
- WRITE_INDEX:
  - Unsigned 32-bit free-running logical slot index.
  - Indicates the position of the next available slot array.

Figure A3.14 summarises the structure of Revere-AMU's ring buffers:

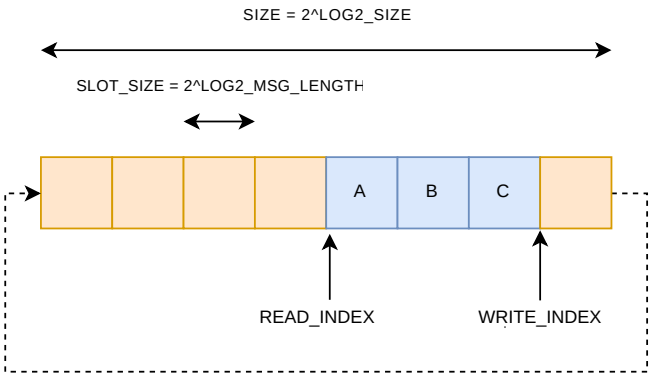

Figure A3.14: Revere-AMU ring structure

A ring is full when (WRITE_INDEX - READ_INDEX) = 2^{LOG2_SIZE}.

A ring is empty when WRITE_INDEX = READ_INDEX.

The number of used slots is (WRITE_INDEX - READ_INDEX).

The number of empty slots is (2^{LOG2_SIZE} - (WRITE_INDEX - READ_INDEX)).

For an TX AMS, the software is a producer and AMU is a consumer.

For an RX AMS, the software is a consumer and AMU is a producer.

For AMI-SW to AMI-SW communications, the SLOT_SIZE of the RX socket (AMS) ring shall be greater or equal to the SLOT_SIZE of the TX AMS ring buffer.

A message shall not span across multiple slots.

Consumer checks if new entries are available in the ring buffer, reads new messages from the slot array as indicated by READ_INDEX and increments this index.

Producer checks if empty slots are available in the ring buffer, writes new messages to the slot array as indicated by WRITE_INDEX and increments this index.

Receive ring buffers have two possible operating modes: the back-pressure and the overwriting modes.

A3.7.2.2.1 Receive operating modes

Back-pressure operating mode

When the back-pressure mode is selected, the producer (AMU for a RX AMS and a software for a TX AMS) must check that empty slots are available.

If slots are available, the producer writes a message at the next available slot in ring identified by WRITE_INDEX and increments this index.

Listing A3.1: Enqueue in back-pressure mode

```
if((WRITE_INDEX - READ_INDEX) == (1 << LOG2_SIZE)
  return (RETRY_LATER);
RING_BASE_PTR[ (WRITE_INDEX & MASK) << (LOG2_MSG_LENGTH + 3)] = Message;
WRITE_INDEX++;
```

Overwriting operating mode

When the overwriting mode is selected, the producer checks whether empty slots are available.

If slots are available, the producer writes a message at the next available slot in ring identified by WRITE_INDEX and increments this index.

If there are no free slots available, the producer:

- Attempts to increment READ_INDEX.
- Writes a message at the next available slot in ring identified by WRITE_INDEX.
- Increments WRITE_INDEX.

This operating mode requires the producer to check whether the ring is full or not and update READ_INDEX atomically. This can be achieved through a compare and swap operation if indices are stored in memory, or by monitoring writes to READ_INDEX if indices are stored in registers. The following code snippet abstracts this operation through COMPARE_AND_SWAP(param1, param2, param3), where:

- param1 refers to the physical location of READ_INDEX.
- param2 refers to the contents of READ_INDEX as observed by the producer at the time it checks for queue fullness.
- param3 refers to the value that the producer is to store in the physical location of READ_INDEX.

Listing A3.2: Enqueue in overwriting mode

```
if((WRITE_INDEX - READ_INDEX) == (1 << LOG2_SIZE)
  COMPARE_AND_SWAP(&READ_INDEX, READ_INDEX, READ_INDEX + 1)
RING_BASE_PTR[ (WRITE_INDEX & MASK) << (LOG2_MSG_LENGTH + 3)] = Message;
WRITE_INDEX++;
```

The compare and swap operation may fail if the consumer updates READ_INDEX after the producer has read it from memory but before the producer updates it. In this case, the producer must:

- Ignore this condition if the number of messages to be enqueued is one, since the producer is guaranteed to have at least one slot available.
- Re-try the operation otherwise.

A3.7.3 Digests

AMU maintains status bits that indicate if new messages are available on RX sockets (AMSs) and if space is available on TX sockets (AMSs).

All the TX status bits of a given AMI-SW are grouped into TX_DIGEST.

All the RX status bits of a given AMI-SW are grouped into RX_DIGEST.

There is one TX_DIGEST and one RX_DIGEST per AMI-SW.

Those status bits are updated based on ring write and read indices, independently from AMI functional state, but are not mandated to always reflect the sockets state.

A3.7.3.1 Threshold

Each socket (AMS) includes a Threshold for controlling the change of associated DIGEST bit. This threshold can be expressed with fixed or relative values. Relative values of the Threshold are expressed in fraction of the number of slots in the ring.

When Threshold or more messages are available in RX AMS #n, the AMU sets the bit #n to '1' in RX_DIGEST.

When less than Threshold messages are available in a RX AMS #n, the AMU clears the bit #n in RX_DIGEST.

When Threshold or more slots are available in TX AMS #n, the AMU sets the bit #n to '1' in TX_DIGEST.

When less than Threshold slots are available in a TX AMS #n, the AMU clears the bit #n in TX_DIGEST.

Except for values 0 and 15, Threshold is expressed in number of 16th of slots and the equation for Threshold is:

$Threshold = \frac{THRESHOLD * slots}{16}$

Where slots is the total number of slots in the ring. The result of the division by 16 is truncated to obtain an integer.

Table A3.37 details how the Threshold value is encoded in the THRESHOLD parameter:

Table A3.37: Encoding of the Threshold value in THRESHOLD.

THRESHOLD value	Corresponding Threshold
0	1
1	$\frac{1}{16} slots$
2	$\frac{1}{8} slots$
3	$\frac{3}{16} slots$
4	$\frac{1}{4} slots$
5	$\frac{5}{16} slots$
6	$\frac{3}{8} slots$

THRESHOLD value	Corresponding Threshold
7	$\frac{7}{16} slots$
8	$\frac{1}{2} slots$
9	$\frac{9}{16} slots$
10	$\frac{5}{8} slots$
11	$\frac{11}{16} slots$
12	$\frac{3}{4} slots$
13	$\frac{13}{16} slots$
14	$\frac{7}{8} slots$
15	slots

The special value zero for THRESHOLD corresponds to an absolute value of exactly one slot.

The special value 15 for THRESHOLD corresponds to slots, the total number of slots in the ring.

A3.7.3.2 RX_DIGEST

Read-only

- [63:0] Each bit corresponds to the status of a RX socket.
  - "1" in bit #n means that Threshold or more messages are available in the ring n (e.g (WRITE_INDEX - READ_INDEX) >= Threshold)
  - "0" in bit #n means that the RX ring #n has less than Threshold messages (e.g (WRITE_INDEX - READ_INDEX) < Threshold)

When the Revere-AMU implements AMI type A1 or A2, this is a register. When the Revere-AMU implements AMI type B, this is a memory location.

A3.7.3.3 TX_DIGEST

Read-only

- [63:0] Each bit corresponds to the status of a TX socket.
  - "1" in bit #n means that more empty slots than Threshold are available in the ring #n (e.g (2^{LOG2_SIZE} - (WRITE_INDEX - READ_INDEX)) >= Threshold)
  - "0" in the bit #n means that the TX ring #n has less than Threshold empty slots (e.g (2^{LOG2_SIZE} - (WRITE_INDEX - READ_INDEX)) < Threshold)

When the Revere-AMU implements AMI type A1 or A2, this is a register. When the Revere-AMU implements AMI type B, this is a memory location.

A3.7.4 AMS Interrupt Masking and Control

Interrupts for an AMI-SW can be masked on a per-AMS basis and can also be enabled globally for TX and RX.

Each AMI-SW has two corresponding interrupt vectors configured by management messages (see A3.4.4.1.2 *AMI-SW Configuration*).

When the Revere-AMU implementation supports interrupts:

- Each AMI-SW has an RX_DIGEST_MASK, which allows to enable or disable the generation of the associated interrupt vector for RX on a per-RX_AMS basis.
- Each AMI-SW has a TX_DIGEST_MASK, which allows to enable or disable the generation of the associated interrupt vector for TX on a per-TX_AMS basis.
- Each AMI-SW has two parameters (TX_IRQ_ENABLE and a RX_IRQ_ENABLE) configured through management messages A4.2.3 *AMI-SW Management Commands and Responses*, which allows to enable or disable the generation of the associated interrupt vector associated with an AMI-SW.

The conditions for an AMI-SW to generate an interrupt are detailed in A3.7.7 *Interrupts*.

A3.7.4.1 RX_DIGEST_MASK

- [63:0] Each bit corresponds to the interrupt mask of a RX socket.
  - "1" in bit #n means that RX AMS #n interrupt generation is masked and can therefore not generate an interrupt
  - "0" in bit #n means that RX AMS #n may generate an interrupt (and is also subject to the RX_IRQ_ENABLE parameter)

This register or memory location provides control over the generation of interrupts on a per RX AMS basis. Changing its value can generate an interrupt under circumstances described in A3.7.7 *Interrupts*.

When the Revere-AMU implements AMI type A1 or A2, this is a register. When the Revere-AMU implementation does not support interrupts, this register is reserved, RES0. Otherwise its reset value is 0xffffffffffffffff (all RX interrupts masked).

When the Revere-AMU implements AMI type B, this is a memory location. When the Revere-AMU implementation does not support interrupts, this memory location is reserved.

A3.7.4.2 TX_DIGEST_MASK

- [63:0] Each bit corresponds to the interrupt mask of a TX socket.
  - "1" in bit #n means that TX AMS #n interrupt generation is masked and can therefore not generate an interrupt
  - "0" in bit #n means that TX AMS #n may generate an interrupt (and is also subject to the TX_IRQ_ENABLE parameter)

This register or memory location provides control over the generation of interrupts on a per TX AMS basis. Changing its value can generate an interrupt under circumstances described in A3.7.7 *Interrupts*.

When the Revere-AMU implements AMI type A1 or A2, this is a register. When the Revere-AMU implementation does not support interrupts, this register is reserved, RES0. Otherwise its reset value is 0xffffffffffffffff (all TX interrupts masked).

When the Revere-AMU implements AMI type B, this is a memory location. When the Revere-AMU implementation does not support interrupts, this memory location is reserved.

A3.7.5 AMI-SW types

The AMS data structures contain information (such as the digests and ring buffer indices) which are frequently updated by the software and the AMU.

Revere-AMU System Architecture can be implemented to operate with different AMI-SW types that:

- provide guarantees to software
- define methods for exchanging fields of the AMS data structure between software and AMU
- prescribe the location of each data structure field, either in Normal memory or a memory mapped register.

The AMI-SW type implemented is discoverable by accessing the AMU_IDR in the memory-mapped management registers.

Type A – Some fields of the AMS data structure are located in memory mapped registers internal to the AMU, other fields are in Normal memory.

Type A1 – The view of memory mapped registers is not consistent, for any PE in the system. Memory mapped register addresses might be aliased, so that each address can map to multiple locations in the system.

Note

This allows implementations using a local (for example: per-PE or per-cluster) peripheral to optimize performance. This AMI type typically requires pinning of software threads.

Type A2 – The view of memory mapped registers is consistent, for any PE in the system. Each memory mapped register address maps to a single physical register in the system.

Note

This can be achieved by a single physical peripheral that implements registers in a given address range, or multiple peripherals that alias the same address range, but implement a coherence mechanism.

Type B – The entirety of the AMS data structure is allocated in Normal memory.

A3.7.5.1 AMI-SW of type A1 and A2

When AMI type A1 or A2 is implemented, the AMS data structures are implemented in the AMU as memory mapped registers and exposed to software through the AMI-SW in the Functions BAR Memory Space aperture.

The layout of the memory-mapped registers is described in Figure A3.15.

All non-reserved fields are R/W except TX_DIGEST and RX_DIGEST which are Read-only.

All fields reset to zero except TX_DIGEST_MASK and RX_DIGEST_MASK which reset to 0xffffffff.

All Reserved fields are RES0.

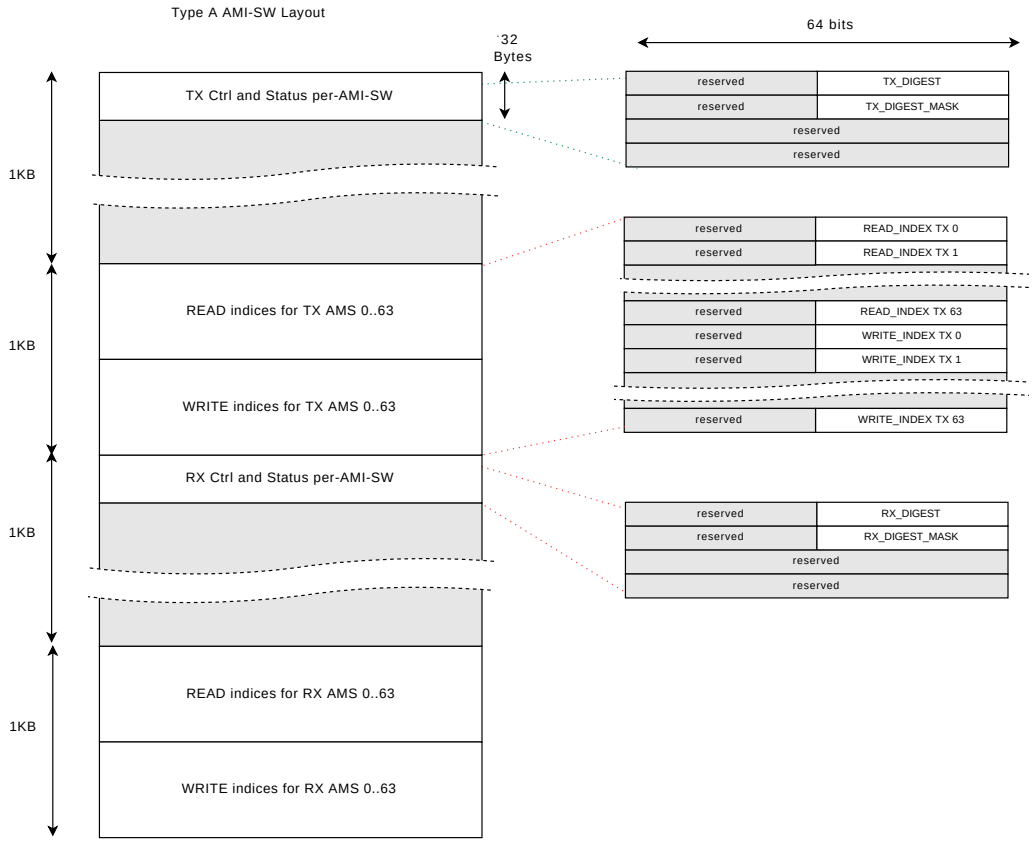

Figure A3.15: Layout of the AMI-SW for Type A

A3.7.5.2 Example of operations for an AMI type A1 or A2

Assumptions is that software maintains a copy of the AMS data structure in coherent shared memory in order to improve the performance (especially for all accesses to the information which is persistent to the duration of an established ASNs).

The following section describes all the steps that would happen when a software sends a message through TX AMS #s of AMI #i.

1. Using all the parameters of the ring buffer,
2. The software determines if some space is left in the ring by accessing the TX_DIGEST.
3. The software stores a message in the ring buffer.
4. The software issues a `DSB ST` barrier instruction to guarantee that the write of the message is globally observable.
5. The software increments its WRITE_INDEX register (in its AMS data structure).
6. AMU detects an update to the write index of socket #s, interface #i.

Note

How Revere-AMU actually achieves this detection is implementation dependent. For example, Revere-AMU could just poll the WRITE_INDEX register or the WRITE_INDEX register could act as doorbell that triggers some actions.

---

7. AMU determines if the destination is an AMI-SW or AMI-HW and the destination RX socket (AMS).
8. AMU determines if the message can be written to the destination RX socket (AMS).

   a) If the destination is part of an AMI-SW, the AMU checks the available space by comparing the read and write indices of the destination RX AMS.
   b) If the destination is part of an AMI-HW, the AMU uses the hardware control flow scheme.
9. AMU reads the message from the TX socket (AMS) ring buffer and forwards it to the destination RX socket (AMS).
   a) If the destination is part of an AMI-SW, The AMU copies the message to the destination ring buffer.
   b) If the destination is part of an AMI-HW, the AMU forwards the message to the destination RX_AMS using the pin-level interface.

The following describes all the steps that would happen when a software receives a message through RX socket (AMS) #s of interface (AMI) #i.

1. The software determines that some data is available from RX AMS #s of AMI #i by accessing the RX_DIGEST of AMI #i.
2. The software accesses the AMS data structure with indices (i,s) to get all the parameters of the ring buffer.
3. The software reads one (or multiple) message(s) from the ring buffer at index READ_INDEX.
4. The software increments its local READ_INDEX and writes this updated value to the memory mapped register.
5. AMU detects an update to the READ_INDEX of AMS #s, AMI #i.

Note

How Revere-AMU actually achieves this detection is implementation dependent. For example, Revere-AMU could just poll the READ_INDEX register or the READ_INDEX register could act as doorbell that triggers some actions.

The following describes all the steps that would happen when software receives a message through RX socket (AMS) #s of interface (AMI) #i, when the socket is operating in overwriting mode. Software is assumed to keep an updated local copy of the read index (local_read_index) for the purposes of detecting lost messages.

1. Software determines that some data is available from RX AMS #s of AMI #i by accessing the RX_DIGEST of AMI #i.
2. Software accesses the AMS data structure with indices (i,s) to get all the parameters of the ring buffer.
3. Software reads one (or multiple) message(s) from the ring buffer at index READ_INDEX.
4. Software updates the read index in the AMS data structure.
5. AMU detects an update to the read index of AMS #s, AMI #i.
6. Software may now calculate the number of messages that were overwritten by the AMU using the following formula: n_messages_lost = READ_INDEX - local_read_index.
7. Software updates local_read_index to the new value.

A3.7.5.3 AMI-SW of type B

When the AMU is implemented to operate with Type B AMI-SW, the AMU is state-less and can be fully distributed.

This type of AMI-SW would typically be used in environments that require software thread migrations.

The AMS data structures (digests, digest masks and ring indices) are arranged in the architected TYPEB_AMI_SW table allocated in Normal memory.

Each AMI-SW has one TYPEB_AMI_SW table. The base address (TYPEB_TBL_BASE_PTR) is configured through the architected messages PF-AMI-SW-CONFIGURE and F-AMI-SW-CONFIGURE and is aligned to 4KB. On a fault when the AMU accesses this pointer, a Function error is triggered (see A3.11.2.2 *Run-time errors*).

Memory accesses performed by the AMU to the TYPEB_AMI_SW table are translated by an SMMU according to the translation scheme assigned to the Function, to which the AMI-SW is mapped. When the Revere-AMU

implementation supports PASID, accesses to the TYPEB_AMI_SW table can use a PASID, as configured in the AMI-SW.

The size of the TYPEB_AMI_SW table is fixed and equal to 4KB.

The layout of the TYPEB_AMI_SW table is similar to the one used in Figure A3.15.

A set of doorbell memory-mapped registers can be used to notify the AMU that updates have been done to the TYPEB_AMI_SW table.

Notifications of new messages (or batch of new messages) are realized through writes to TX_DB_x doorbell registers. Notifications of consumed messages (or batch of new messages) are realized through writes to RX_DB_x doorbell registers.

The AMI-SW contains one doorbell register per AMS.

The layout of the memory-mapped registers is described in Table A3.38.

Table A3.38: AMI-SW for type B.

Offset	Bits	Name	Description
+ TBD	[63:0]	reserved	RES0
...			
+ TBD	[63:0]	TX_DB_0	Doorbell register for TX AMS #0
+ TBD	[63:0]	TX_DB_1	Doorbell register for TX AMS #1
...			
+ TBD	[63:0]	TX_DB_63	Doorbell register for TX AMS #63
...	[63:0]	reserved	RES0
+ TBD	[63:0]	RX_DB_0	Doorbell register for RX AMS #0
+ TBD	[63:0]	RX_DB_1	Doorbell register for RX AMS #1
...			
+ TBD	[63:0]	RX_DB_63	Doorbell register for RX AMS #63

A3.7.5.3.1 Doorbells

Each doorbell register follows the following format:

Write-only, Read-As-Zero.

Table A3.39: TX_DB and RX_DB format

Offset	Bits	Name	Description
0x0	[63:32]	Reserved	RES0
0x0	[31:0]	ELEM_CNT	Number of element written to the ring buffer.

A3.7.5.4 Example of operations for an AMI of type B

The following section describes all the steps that would happen when a software sends a message through TX AMS #s of AMI #i.

1. The software accesses all the parameters of the ring buffer.
2. The software determines if some space is left in the ring by accessing the TX_DIGEST.
3. The software stores a message in the ring buffer at index WRITE_INDEX.
4. The software increments its write index (in its TYPEB_AMI_SW table)
5. The software issues a `DSB ST` barrier instruction to guarantee that the write of the WRITE_INDEX and all preceding writes to the ring and out-of-band buffers are globally observable.
6. The software notifies AMU by writing the number of slots it just enqueued to the doorbell registers of the AMS #s.
7. AMU detects an update to the doorbell of socket (AMS) #s, interface (AMI) #i.
8. AMU determines if the destination is an AMI-SW or AMI-HW and the destination RX socket (AMS).
9. AMU determines if the message can be copied to the destination RX socket (AMS).
   a) If the destination is part of an AMI-SW, the AMU checks the available space by comparing the read and write indices of the destination RX socket (AMS).
   b) If the destination is part of an AMI-HW, the AMU uses the hardware control flow scheme.
10. AMU reads the message from the TX AMS ring buffer and forwards it to the destination RX AMS.
    a) If the destination is part of an AMI-SW, the AMU writes the message to the destination ring buffer.
    b) If the destination is part of an AMI-HW, the AMU forwards the message to the destination RX AMS using the pin-level interface.

The following describes all the steps that would happen when a software receives a message through RX socket (AMS) #s of interface (AMI) #i.

1. The software determines that some data is available from RX AMS #s of AMI #i by accessing the RX_DIGEST of AMI #i.
2. The software accesses all the parameters of the ring buffer.
3. The software reads one (or multiple) message(s) from the ring buffer at index READ_INDEX.
4. The software increments the READ_INDEX.
5. The software issues a `DSB ST` barrier instruction to guarantee that the write to the READ_INDEX is globally observable.
6. and notifies AMU by writing the number of element it just dequeues to the doorbell registers of the AMS #s.
7. AMU detects an update to doorbell of socket (AMS) #s, interface (AMI) #i.
8. AMU initiates a DMA read transactions to get the updated value of the READ_INDEX.

The following describes all the steps that would happen when software receives a message through RX socket (AMS) #s of interface (AMI) #i, when the socket is operating in overwriting mode. Software is assumed to keep an updated local copy of the read index (local_read_index) for the purposes of detecting lost messages.

1. Software determines that some data is available from RX AMS #s of AMI #i by accessing the RX_DIGEST of AMI #i.
2. Software accesses the TYPEB_AMI_SW table with indices (i,s) to get all the parameters of the ring buffer.
3. Software reads one (or multiple) message(s) from the ring buffer at index READ_INDEX.
4. Software attempts to update the read index in the AMS data structure by issuing a compare and swap operation.
   a) If the update succeeds, software continues to step 5.
   b) If the update fails, software discards any message(s) read and goes back to step 2.
5. Software issues a `DSB ST` barrier instruction to guarantee that the write to the READ_INDEX is globally observable.
6. Software notifies AMU by writing the number of element it just dequeued to the doorbell registers of the AMS #s.
7. AMU detects an update to doorbell of socket (AMS) #s, interface (AMI) #i.
8. Software may now calculate the number of messages that were overwritten by the AMU using the following formula: n_messages_lost = READ_INDEX - local_read_index.

9. Software updates local_read_index to the new value.

A3.7.6 Ordering requirements

Revere-AMU defines a set of ordering rules for all reads and writes transactions involved in the message send and receive operations with a PE.

Arm ARM terminology (*observers*, *ordered-before*, *out-of-band-ordered-before*, *location*) is being referenced in this section. Please refer to [1] for a complete definition of these terms.

A3.7.6.1 Message send

PE writes to the TX ring buffer slot arrays shall be *out-of-band-ordered-before* reads of the slot array by the AMU.

The enforcement of this ordering is IMPLEMENTATION DEFINED.

For example and for type A, the following sequence of instructions in program order enforces correct ordering:

1. Writes to the TX ring buffer slot array
2. `DSB ST` instruction
3. Write to the WRITE_INDEX

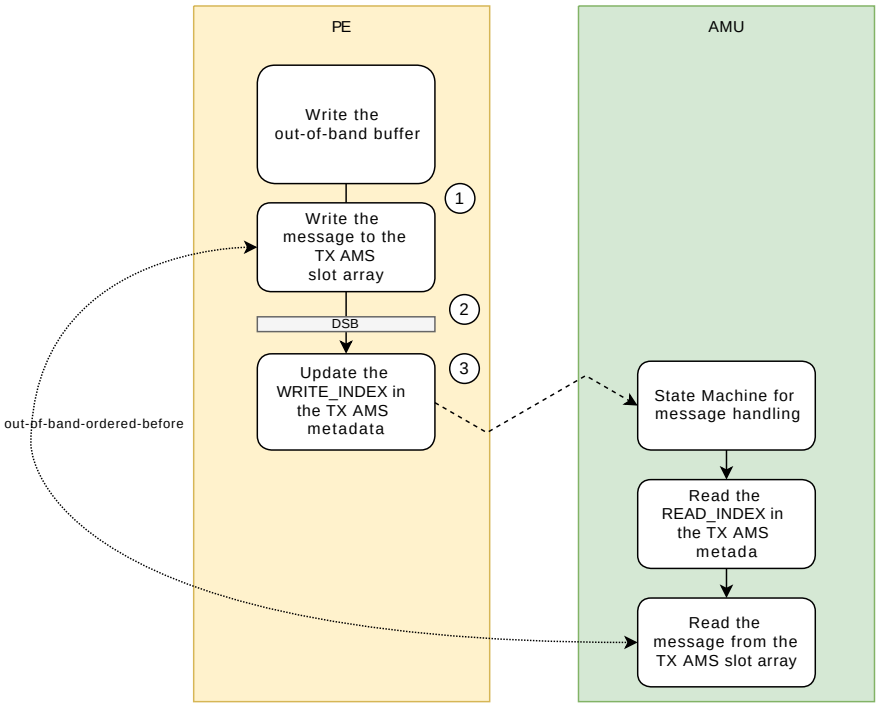

Figure A3.16: AMU Transaction ordering for TX_AMS AMI-SW for Type A.

For type B, the following sequence of instructions in program order enforces correct ordering:

1. Writes to the TX ring buffer slot array
2. Write to the WRITE_INDEX
3. `DSB ST` instruction
4. Write the doorbell register TX_DB

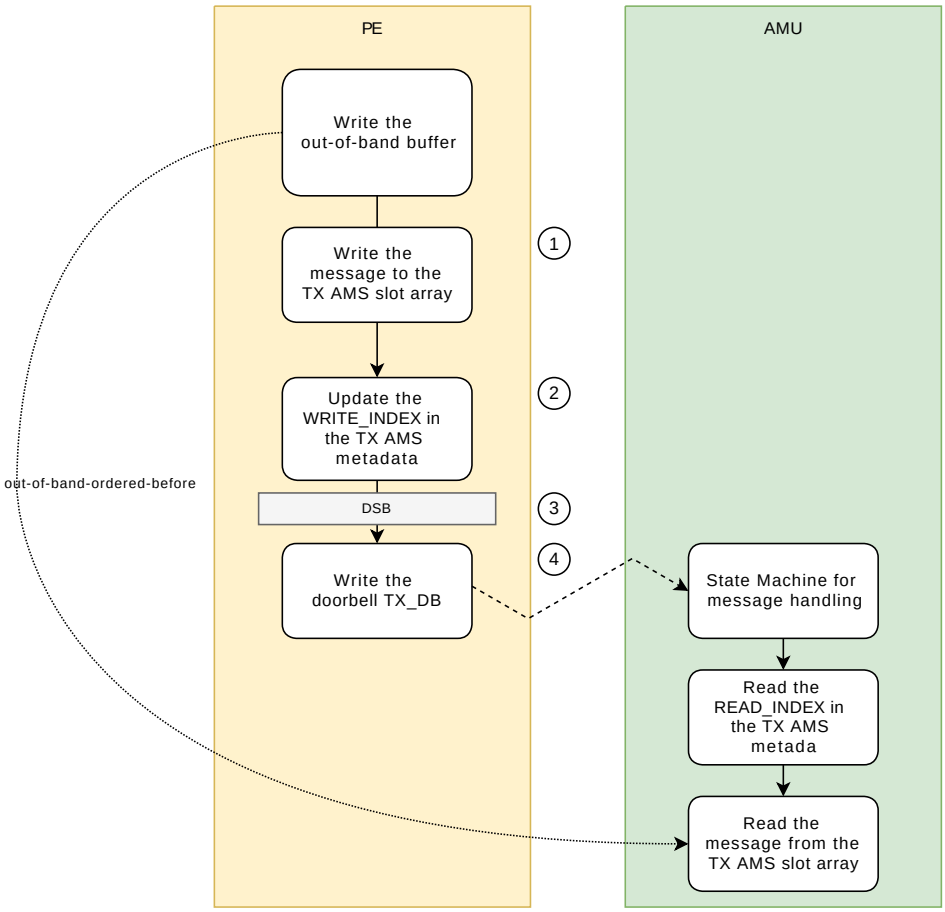

Figure A3.17: AMU Transaction ordering for TX_AMS AMI-SW for Type B.

A3.7.6.2 Message receive

Figure A3.18 describes the sequence of transactions involve when a PE receive a message from an AHA.

Writes to the out-of-band buffer by the AHA shall be *out-of-band-ordered-before* the write of the WRITE_INDEX by the AMU.

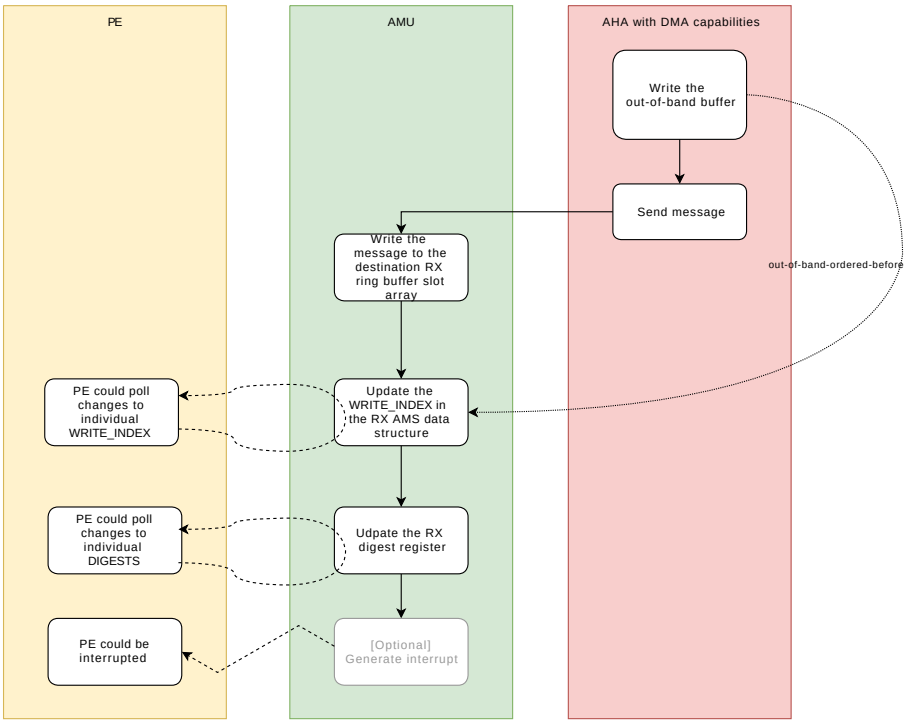

Figure A3.18: AMU Transaction ordering for RX_AMS AMI-SW.

A3.7.6.3 Ordering requirements for Interrupts

Interrupts can be generated after an update to a TX_DIGEST or RX_DIGEST value (see A3.7.7 *Interrupts*).

An interrupt caused by a change of a TX_DIGEST or RX_DIGEST value must be generated only after all PEs in the system can observe the updated value.

A3.7.6.4 Ordering requirements for digests

An update to a ring read or write index usually leads to a digest register update and potentially to an interrupt being generated. TX_DIGEST or RX_DIGEST updated values must be observable only after all PEs in the system can observe the corresponding write/read indices changes.

A3.7.7 Interrupts

This section defines controls that allow software to configure the AMU to generate interrupts.

It is optional for a Revere-AMU Function to support interrupts, and support for interrupts can be detected at runtime by system software (see A3.7.7.4.2 *PCI Capabilities*).

The interrupt mechanisms defined here allow each Revere-AMU Function to:

- Generate an interrupt in response to a change in a TX_DIGEST or RX_DIGEST. For RX socket (AMS) this means that there are pending messages for receive. For a TX socket (AMS) this means that there are pending free slots for transmit.
- Support up to two interrupt vectors per AMI-SW
- Support up to 2048 MSI-X vectors, extensible to close to 2^{31} device specific vectors
- Mask generation of an interrupt per socket (bit of TX_DIGEST and RX_DIGEST)
- In conjunction with per-AMS threshold settings, control per socket (AMS) the minimum number of messages generating an interrupt

- Enable/disable interrupts per AMI-SW
- Dynamically transition between polling and interrupt driven modes without losing events

A3.7.7.1 Overview: Interrupts and Notifications

The Revere-AMU and associated hardware agents (AHAs) can notify a PE of an asynchronous event with a Revere Message.

There are cases however where interrupts may be necessary, for example to notify the PE of the reception of a Revere Message. This allows a software thread running on a PE to wait for Revere Messages without polling.

To notify of the occurrence of an asynchronous event, a Revere-AMU may generate interrupts towards a PE.

When it supports interrupts, a Revere-AMU Function must generate Message Signalled Interrupts (MSI-X) for vectors in 0..2047, which are effectively a write of a 32-bit data to a register address. Vector in 2048..2^{31}-1 are device specific and have an IMPLEMENTATION DEFINED behaviour.

A3.7.7.2 Interrupts sources

The events in a Revere Device causing interrupts are related to an AMI-SW RX_DIGEST or TX_DIGEST change (or to error reporting, see A3.11.2.1 *Configuration-time errors*).

The following events in an AMI-SW can generate an interrupt:

- Reception of more messages than a programmed threshold in an RX socket (AMS), effectively changing the RX_DIGEST value
- Freeing of more messages than a programmed threshold by the AMU in a TX socket (AMS), effectively changing the TX_DIGEST value

An AMI-SW can generate an interrupt only when it is active (see A3.4.4.6 *AMI and AMS state machines*).

Note

Events originating from PCI Express capabilities, such as the PCI Express Capability or the SR-IOV Capability can also generate interrupts. See the PCI Express Base Specification [2].

In addition to the aforementioned events, the following actions from a PE on a Revere Device may have the side effect of generating an interrupt or delivering a previously pending interrupt:

- A change of the TX_DIGEST_MASK or RX_DIGEST_MASK value
- A change of the TX_IRQ_ENABLE or RX_IRQ_ENABLE parameter value
- A change in PCI MSI-X Table Vector Control Register Mask Bit, Function Mask or Command Register Bus Master Enable (BME)
- A change in IMPLEMENTATION DEFINED configuration

A3.7.7.3 Configuration

Generation of interrupts for a Revere-AMU Function can be controlled through registers and configuration messages sent through the management interface (see A3.4.3 *Management Registers* and A3.4.4 *Management Messages*), through registers in the PCI MSI-X Capability and the BME bit in the PCI Type 0 Header (for MSI-X vectors), and IMPLEMENTATION DEFINED configuration (for device specific vectors).

Threshold can be controlled for each socket (AMS) as explained in A3.7.3.1 *Threshold*.

Interrupts can be masked per socket (AMS) and enabled for TX and RX as explained in A3.7.4 *AMS Interrupt Masking and Control*.

When it supports interrupts, an AMI-SW is able to generate an interrupt vector for TX AMSs and one for RX AMSs. The two interrupt vectors numbers are configured through the management interface (see A3.4.4 *Management Messages*).

TX_VECTOR and RX_VECTOR must be programmed to a valid vector number with the PF-AMI-SW-CONFIGURE or F-AMI-SW-CONFIGURE commands. Only the vector numbers actually available are valid. The availability of MSI-X vectors is determined by the PCI MSI-X Capability Message Control Register Table Size field. The availability of device specific vectors is IMPLEMENTATION DEFINED.

If an interrupt is signalled while its TX_VECTOR or RX_VECTOR is programmed to an invalid vector, an E-F-AMI-RING-INVALID-VECTOR exception is sent by the AMU to the Function in scope.

Note

When multiple AMI-SW TX_VECTOR are programmed to the same value, they effectively share an interrupt vector. When servicing a shared interrupt vector, system software can determine which AMS needs servicing by looking at all the TX_DIGEST values of all the AMI-SW sharing the same vector.

The same remark applies to RX.

Refer to Table A3.40.

A3.7.7.3.1 Logical view of interrupt generation

This section details the logical view of interrupt generation for TX. The corresponding view for RX interrupt generation is obtained by replacing TX by RX for all registers and fields names.

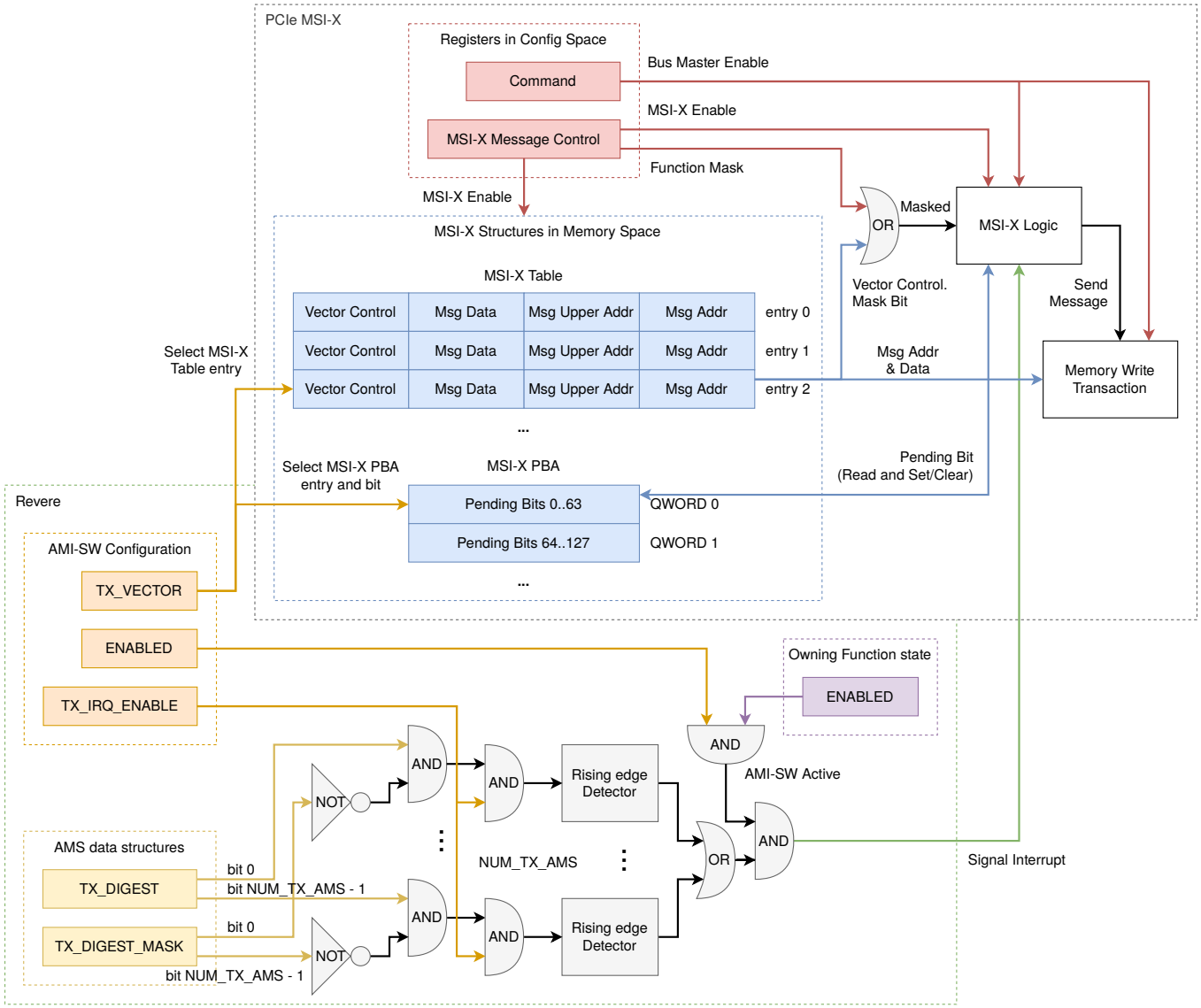

Figure A3.19: Revere Interrupt Generation logical view

Figure A3.19 summarizes the logical process for TX sockets (AMSs) interrupt generation, showing all configuration registers involved (for MSI-X vectors).

Each TX_DIGEST bit n, TX_DIGEST_MASK bit n and the TX_IRQ_ENABLE parameter are evaluated to determine if an interrupt needs to be generated.

Any change in an input, which causes a change of the evaluation from zero to one generates the interrupt vector determined by TX_VECTOR.

For example:

- A change of the TX_DIGEST_MASK value may unmask an AMS condition signalled in the TX_DIGEST and may have the effect of generating an interrupt if enabled by the TX_IRQ_ENABLE parameter
- A change of the TX_IRQ_ENABLE parameter value while TX_DIGEST and TX_DIGEST_MASK signal an interrupt condition may have the effect of generating an interrupt

A3.7.7.4 PCI Express aspects of Interrupts

All Revere-AMU Functions implement the behaviour that is architecturally required by the PCI Express Base Specification [2]. This applies to interrupts.

It is optional for a Revere-AMU Function to support interrupts. When it supports interrupts, a Revere-AMU Function is not permitted to use legacy INTx interrupts or MSI. A Function of a Revere-AMU must either support MSI-X interrupts, or not support interrupts at all.

The write transaction of an interrupt is qualified with the Requester ID of the Revere-AMU Function generating the interrupt vector, with no PASID.

Refer to Table A3.40. Refer to Table A3.45 for details on DMA accesses.

A3.7.7.4.1 PCI Type 0 Header

As it is not allowed to support legacy INTx interrupts, a Function of a Revere-AMU must not use a legacy interrupt Message. Its PCI Type 0 Configuration Space Header must fulfil the following constraints:

- Interrupt Line (Offset 3Ch) must be RO Zero
- Interrupt Pin (Offset 3Dh) must be RO Zero

A3.7.7.4.2 PCI Capabilities

As it is not allowed to support MSI interrupts, a Function of a Revere-AMU must not implement the PCI MSI Capability.

When a Function of a Revere-AMU does not support interrupts, it must not implement a PCI MSI-X Capability, and when a Function of a Revere-AMU supports interrupts, it must implement a PCI MSI-X Capability (see A3.3.2 *PF/VF Configuration Space*).

Note

System software may use the presence of the MSI-X Capability to determine that a Revere-AMU Function supports interrupts.

The PCI MSI-X Table structure and MSI-X Pending Bit Array (PBA) belong to the Function.

When a Revere-AMU Function implements a PCI MSI-X Capability, it must fulfill the following constraints:

- The MSI-X Table and PBA structures must be mapped by a 64-bit prefetchable BAR
- The MSI-X Table and PBA structures can be contained in the same page of 64KB. They must not share a 64KB page with other usable address spaces that are not associated with MSI-X structures

---

Note

Requiring that MSI-X structures do not share a page of 64KB with other address spaces is aligned with the Server Base System Architecture specification [3] requirements.

It is also a performance consideration as it ensures that MSI-X structures can be completely isolated from other Memory Space resources with 64KB page mappings, avoiding the need for system software to “trap and emulate” all accesses to shared pages.

Virtualization scenarios where a hypervisor can “pass-through” the Memory Space of a PCI Express Function to a virtual machine while still isolating MSI-X address spaces can do so with 64KB stage 2 translation mappings with maximum performances.

The following items are also recommended (but not mandatory):

- A Revere-AMU Function should dedicate a BAR to MSI-X structures.
- It is recommended that MSI-X Table and PBA structures of a Revere-AMU Function do not reside in the same page of 64KB.
- The addresses of the first words of the MSI-X structures should be aligned to 64KB.

If the MSI-X structures are mapped in BAR0 they must be placed after the last 64KB page containing AMI-SWs and they must also be mapped after the last 64KB page with IMPLEMENTATION DEFINED contents.

See A3.3.3 *PF/VF BAR Space* for memory space details.

See the PCI Express Base Specification [2] “MSI and MSI-X Capability Structures” and “Single Root I/O Virtualization and Sharing” chapters.

A3.7.7.5 Interrupt vectors summary

Table A3.40 summarizes interrupt vectors in a Revere-AMU implementation:

Table A3.40: Interrupt vectors in a Revere-AMU

Vector number	Description	Behaviour	Availability	Configuration	PASID
0..2047	MSI-X vectors	MSI-X	Determined by MSI-X Capability	MSI-X Capability and structures	None
2048..2^{31}-1	Device specific vectors	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	IMPLEMENTATION DEFINED	None

A3.7.8 Use of Shared Virtual Memory (SVM)

In a Revere-AMU System, the AMU and all associated AHAs can initiate Direct Memory Accesses into process or kernel address spaces.

One way to enable this is having PE and Revere AMU share the same virtual address space.

Some SMMUs (such as SMMUv3 [6]) enable sharing the same page tables between MMU and SMMU, which prevents from having to keep in sync two sets of page tables.

Another benefit from SVM is that it also enables memory over-commit via on demand paging for both PE and device access to memory.

For PCI Express, one possible implementation of SVM uses PCI Express ATS (Address Translation Services), PRI (Page Request Interface) and PASID. See A3.7.9 *Accessing unpinned memory (SMMU page faults)* for ATS and PRI support with SMMUv3 on Revere-AMU systems, and below for PASID support.

note::Implementers should dimension SMMU translation resources according to the number of AMIs supported.

A3.7.8.1 PASID support

When Revere-AMU implements the PASID extended capability, the SMMU may use (RequesterID, PASID, VA) to derive a physical address.

In that case, PASID usage is enabled or disabled at Physical Function level using the PASID Enable flag in the PCI Express PASID Control Register, but a finer grained control is also provided at AMI level (PASID_ENABLE flag). Both flags must be enabled for DMA transactions associated with an AMI to be able to use a PASID qualifier.

The PCI Express standard mandates support for requesting a Function to gracefully stop using a specific PASID. This is achieved through the management commands F-AMI-SW-DISABLE and F-AMI-HW-DISABLE (those commands must be respectively sent to all the AMI-SW and AMI-HW configured with this PASID).

To isolate PCI Express device-private memory in a Revere system (private memory for Revere-AMU or for an AHA) Arm recommends using a PASID in the PF Requester ID space.

Refer to Table A3.45 for details on DMA accesses.

A3.7.9 Accessing unpinned memory (SMMU page faults)

It is optional for a Revere-AMU system to support page faults on its DMA accesses. If supported, then unpinned memory can be used.

In that case, Revere-AMU will rely on the discoverable PRI capability from PCI Express A3.3.2.7 *Page Request Extended Capability* to support page faults.

Such capability requires one or several connections from Revere-AMU to an SMMU implementation (such as SMMUv3 [6]) to convey page requests.

Some implementations might have a closely-coupled SMMU: in such cases there shall be an IMPLEMENTATION DEFINED mechanism to discover if Address Translation Cache invalidations are required as Revere-AMU might then participate in SMMU broadcast TLB invalidation.

Note

To avoid complex hypervisor logic, pinning Stage 2 translation allows direct fault resolution by the guest OS. Similarly for Stage 1. In general, Arm recommends reserving PRI for exclusive Stage 1 or Stage 2 usage.

Note

If a Revere-AMU implements ATS capability A3.3.2.6 *ATS Extended Capability*, AHAs that are not fully trusted still require an SMMU in front of their DMA port to control accesses. ATS support comes with system security implications and implementers are advised to exercise great caution.

Refer to Table A3.45 for details on DMA accesses.

A3.8 Accelerator Message Interface for AHAs

A3.8.1 Overview

This section describes the AMI-HW associated to an AHA context which enables it to send and receive messages.

Contexts running on an AHA exchange messages over established sessions (ASNs) through TX socket (TX AMS) for sending messages and RX socket (RX AMS) for receiving messages.

- AHAs send messages by writing to the logical buffer of a TX socket (TX AMS).
- AHAs receive messages by reading from the logical buffer of a RX socket (RX AMS).

The number of TX socket (NUM_TX_AMS) and the number of RX socket (NUM_RX_AMS) per AMI-HW are IMPLEMENTATION DEFINED and shall be discoverable using the AHA_IIDR field from the PF-PROBE-AHA management command result.

A3.8.2 AHA contexts

When required, AHAs contexts can be configured using an IMPLEMENTATION DEFINED mechanism.

For example:

- AHA could expose configuration registers in the PF BAR with a discoverable offset using the AHA_IIDR field from the PF-PROBE-AHA management command result.
- AHA could have one or several configuration RX AMSes.

Saving and restoring AHA contexts could be achieved using similar IMPLEMENTATION DEFINED mechanisms.

Wherever an AHA context needs to be transferred during live migration (refer to Chapter C2 *VM Live Migration*), such capability is mandatory in order to support this use case.

A3.8.3 AHA AMI-HW Mapping

Each AHA has an IMPLEMENTATION DEFINED number of physical AMI-HW which shall be discoverable using the AHA_IIDR field from the PF-PROBE-AHA management command result.

Those AMI-HW can be mapped to Functions, using a physical AMI-HW-ID which is local to the AHA.

Once mapped, an AMI-HW is assigned a logical AMI-HW-ID which is local to the Function.

To see how those IDs are used in management commands refer to A4.2.4 *AMI-HW Management Commands and Responses*

A3.9 Tracing

A3.9.1 Overview

This section describes the tracing mechanisms that are provided to aid the development of software as well as to enable post-mortem analysis of applications. This is an optional feature of the architecture.

The Revere-AMU architected tracing features include facilities to:

- Obtain a trace of messages between two sockets (AMSs) in the system, configurable on a per-session (per-ASN) basis.
- Deliver traces directly to any Function driver in the system.
- Trace the entirety of a message or just metadata associated with the message.
- Capture all messages or best-effort to the extent that the receiving driver can keep up.

The AMU can generate a message trace as a result of a message being sent from one socket (AMS) to another socket (AMS), irrespective of whether both or either of the endpoints belong to an AMI-SW or an AMI-HW. Whether a message trace is generated and the actual contents depends on how Function drivers configure the system (See A3.9.5 *Tracing Configuration*).

Traces in a Revere-AMU system are transported as messages through sockets in the management AMI. A trace is composed of metadata associated with the message that was sent and (optionally) the original message itself.

A3.9.2 ASN Monitoring Ownership

Message traces are always associated with a session (ASN) and are sent to exactly one consumer, referred to as the ASN monitoring owner. The ASN monitoring owner can be any Function driver.

The ASN monitoring owner is configured by the PF at ASN creation time through the management interface. The monitoring owner can be any Function driver in the system, when such Function driver is not within the scope of that ASN. A monitoring owner of 0 will set the PF driver as the owner. Multiple ASN with the same owner get aggregated into the same trace AMS.

Note

A monitoring owner is also responsible for profiling (See A3.10.2 *ASN profiling functionality*). This means that both traces and counter overflow notifications associated with an ASN will be delivered to the monitoring owner.

A3.9.3 Trace transport and storage

Revere-AMU leverages the same message-passing mechanism it provides for trace transport and storage; traces are transported as messages and sent to a trace socket (AMS), which effectively acts as a trace buffer.

The AMU acts as an aggregator for all traces. In particular, the AMU:

- Generates message traces when a message is being transported between two sockets (AMSs).
- Optionally, adds the original message to the trace.
- Determines the session (ASN) monitoring owner.
- Sends generated traces to a trace socket in the management AMI of the ASN monitoring owner.

The following diagram shows the tracing data path:

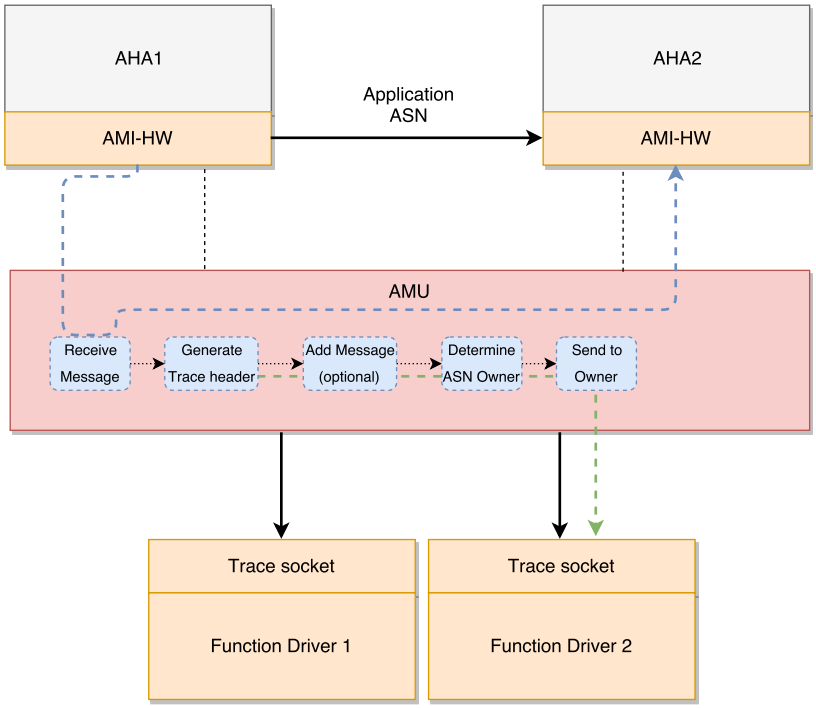

Figure A3.20: A message is sent from AHA1 to AHA2 and, as a result, a trace message is generated by the AMU and sent to the ASN monitoring owner through a trace socket. The blue line depicts the original message path, and the green line depicts the trace message path.

Once traces reach the RX socket (AMS), the session (ASN) monitoring owner can dequeue them from the ring. Since traces can contain the original message, they have a variable size. Software can determine the size of the original message by subtracting the trace header size from the message size.

Note

Tracing often requires that old traces are overwritten with new traces when the trace buffer is full. This behaviour can be implemented by configuring the RX socket (AMS) to operate in overwriting mode (See A3.7 *Accelerator Message Interface for Software*).

A3.9.4 Trace format

The trace available to the software driver from the trace session (ASN) contains:

- A trace header, which contains metadata associated with the original message.
- Optionally, the original message.

The following table captures the format of the trace message:

Offset	Bits	Name	Description
+0x00	[31:0]	- Reserved -	Reserved, RES0
+0x04	[31:16]	SRC_F_OWNER	Function that owns the sending AMI
+0x04	[15:0]	SRC_AMI	Sending AMI within SRC_F_OWNER
+0x08	[31:16]	DST_F_OWNER	Function that owns the receiving AMI
+0x08	[15:0]	DST_AMI	Receiving AMI within DST_F_OWNER
+0x0C	[31:14]	- Reserved -	Reserved, RES0

Offset	Bits	Name	Description
+0x0C	[13]	SRC_AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x0C	[12]	DST_AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x0C	[11:6]	SRC_AMS	Sending AMS within SRC_AMI
+0x0C	[5:0]	DST_AMS	Receiving AMS within DST_AMI
+0x10	[63:0]	TIMESTAMP	Global timestamp value
+0x18		MESSAGE	Original message (arbitrary length)

Note

Out-of-band data is not captured and traced by the mechanisms that the Revere-AMU architecture provides. If this is required by a particular use-case, the application must ensure that pointers in the original message can be dereferenced by the ASN monitoring owner and that it is safe to do so when the trace is consumed.

Note

The application must ensure that the slot size of the RX socket (AMS) that is used for tracing is equal or bigger than the maximum slot size used across all of the session (ASN) traced within the scope of this RX socket (AMS) plus the size of the architected trace header. This is to guarantee that a trace message will always fit within the slot.

A3.9.5 Tracing Configuration

Software can determine whether an implementation supports tracing by checking AMU_IDR.TRACING. When this bit is set to 1, the implementation supports tracing.

Tracing is configured through the following controls:

- AMU_CR register. This register contains a per-AMU control (available to the PF) and a per-Function control (available to all Functions).
- MON_OWNER and TRACE_CTL, controlled through the management interface (See A4.2.6 *AMS Management Commands and Responses*). These are per-ASN controls.

The TRACE_CTL field is a single 16-bit half word, and contains the following controls:

- [15:3] Reserved
- [2] INC_MSG
- [1] EXT_TRACING
- [0] EN

For tracing to be enabled for a particular session (ASN), the following conditions must be met:

- If the consumer is a VF driver, AMU_CR.VF_TRACE_EN is set to 1.
- AMU_CR.TRACE_EN is set to 1.
- TRACE_CTL.EN is set to 1.

If the above conditions are met, the ASN monitoring owner will receive any message traces generated within the scope of that particular ASN. The ASN monitoring owner is configured through the management interface.

If AMU_CR.VF_TRACE_EN == 0, the AMU must not deliver any traces to VF drivers. Similarly, if AMU_CR.TRACE_EN == 0 in a given Function, the AMU must not deliver any traces to that Function. If a given

ASN has TRACE_CTL.EN == 0, the AMU must not generate any traces that are within the scope of that ASN.

Message traces may include the original message. This is done through TRACE_CTL.INC_MSG. When this bit is set to 1, the original message is appended to the trace header.

The EXT_TRACING bit can be used to enable an IMPLEMENTATION DEFINED distribution tracing infrastructure (See A3.9.7 *Implementation defined tracing*). When this bit is set to 1, the IMPLEMENTATION DEFINED distribution tracing infrastructure is enabled for this ASN and the MON_OWNER field is ignored for the purposes of tracing.

Note

Function drivers receive traces through their Trace socket in their management AMIs (See A3.4 *Management Interface*).

A3.9.6 Ordering and crediting implications

When tracing is enabled the AMU does not retrieve the original message from the TX socket (AMS) unless buffer space is available at both the destination RX socket (AMS) and the RX AMS that is part of the management AMI. This is to prevent issues where a trace is delivered and processed by the ASN monitoring owner even though the original message was not delivered to the consumer, or where a message is delivered but not traced.

Note

This means that a trace AMS with no remaining credits can block the application session (ASN) indefinitely. If this is unacceptable for a given application, the trace AMS should be configured in overwriting mode (See A3.7 *Accelerator Message Interface for Software*).

A3.9.7 Implementation defined tracing

An implementation may in addition support IMPLEMENTATION DEFINED distribution tracing infrastructure for self-hosted or non self-hosted debugging. If the IMPLEMENTATION DEFINED distribution tracing infrastructure supports capturing the Revere-AMU traces, then trace messages within the scope of a session (ASN) with EXT_TRACING == 1 should be routed to the IMPLEMENTATION DEFINED infrastructure.

Note

Coresight is an example of IMPLEMENTATION DEFINED tracing infrastructure.

A3.10 Profiling

A3.10.1 Overview

This section describes the profiling mechanisms that Revere-AMU provides. This is an optional feature of the architecture.

Session (ASN) profiling enables per-ASN performance monitoring through:

- A set of architected counters backed by a block of normal memory associated with an ASN.
- A mechanism to notify the PF driver or a VF driver when a counter overflows.
- Controls to enable/disable profiling on a per-ASN basis.
- Controls to enable/disable counter overflow notifications on a per-ASN and per-counter basis.

Session (ASN) counters are always associated with an event type, which describes what is being counted. There is a one to one mapping between a counter and an event type.

Counter overflow notifications are received through the management AMI.

A3.10.2 ASN profiling functionality

Session (ASN) profiling enables performance monitoring on a per-ASN basis through a maximum of 64 counters exposed to software through a block of memory associated with a particular ASN called ASN profiling table. The ASN profiling table is allocated by a Function driver.

Counters count events, and these can trigger at any point throughout the system. The Revere-AMU architecture specifies a set of architected ASN event types. Part of the event ID space is reserved for IMPLEMENTATION DEFINED event types (See A3.10.3 *ASN profiling event types*).

If the feature is enabled, counters are incremented by 1 when the associated event occurs and system software can read them at any point in time. Session (ASN) counters are stored in an ASN profiling table, configured through the management interface (See A4.2.6 *AMS Management Commands and Responses*). An ASN has a monitoring owner. A monitoring owner is always a Function driver, even if such Function driver is not within the scope of the ASN.

Note

A monitoring owner is also responsible for tracing (See A3.9.2 *ASN Monitoring Ownership*). This means that both counter overflow notifications and traces associated with a session (ASN) will be delivered to the monitoring owner.

Reset value is 0 for all of the ASN counters. When a counter overflows the AMU can generate a counter overflow notification message. This notification is sent through the exception socket in the management AMI (See A3.10.6 *Counter overflow notifications*).

A3.10.3 ASN profiling event types

Session (ASN) events are indicated by a 6-bit numeric ID, in the following ranges:

- 0x00 to 0x1F: Architected events
- 0x20 to 0x3F: IMPLEMENTATION DEFINED events

The architecture defines the following event types and their causes:

Event ID	Description	Offset in ASN profiling table	Notes
0	Message processed	0x00	A message has been dequeued from the TX AMS and delivered to the RX AMS.
1	Byte transmitted	0x08	One byte has been read from the TX AMS and written to an RX AMS and the associated message has been delivered to the RX AMS. This applies to both the message header and any in-band data that might be included in the message.
2	Stalled cycle due to lack of credits	0x10	A message intended for this RX AMS cannot be delivered during this cycle due to lack of credits. The period between cycles is IMPLEMENTATION DEFINED.
3	TX AMS digest change	0x18	A message has been dequeued from the TX AMS and, as a result, the number of empty slots has reached TX_THRESHOLD and the associated digest bit has been set to 1.
4	RX AMS digest change	0x20	A message has been enqueued to the RX AMS and, as a result, the number of filled slots has reached RX_THRESHOLD and the associated digest bit has been set to 1.
5	Out-of-band stash request	0x28	A dataless out-of-band stash request has been generated by the AMU as a result of a message being sent through this ASN.

Software can discover the number of ASN event types supported by sending an F-PROBE command through the management AMI.

The value in ASN_PROF_N_COUNTERS includes both all the architected counters and any IMPLEMENTATION DEFINED counters.

A3.10.4 ASN profiling Configuration

Software can determine whether an implementation supports ASN profiling by checking AMU_IDR.ASN_PROF. When this bit is set to 1, the implementation supports ASN profiling. Software can determine the number of ASN counters an implementation supports by sending an F-PROBE command through the management interface.

ASN profiling is configured through the following controls:

- AMU_CR register. This is a per-Function control.

- PROF_CTL, PROF_MASK and PROF_TBL_BASE_PTR, controlled through the management interface (See A4.2.6 *AMS Management Commands and Responses*). These are per-ASN controls.

PROF_CTL is a single 16-bit half word, and contains the following controls:

- [15:1] Reserved
- [0] EN

For ASN profiling to be enabled for a particular ASN, the following conditions must be met:

- AMU_CR.ASN_PROF_EN is set to 1.
- PROF_CTL.EN is set to 1.

If the above conditions are met the AMU will count events associated with this ASN.

Software can configure PROF_MASK to enable or disable counter overflow notifications for each counter independently. When bit N is set to 1, the AMU sends a notification to the ASN monitoring owner through the exception AMS in the management AMI (See A5.2.2 *Monitoring Exceptions*).

PROF_TBL_BASE_PTR must be configured with an address pointing to an ASN profiling table and shall be Doubleword aligned. This address must be accessible by the monitoring owner. This field must only be changed when ASN profiling is disabled for this ASN.

On a fault when the AMU accesses this pointer, a Function error is triggered in the monitoring owner (see A3.11.2.2 *Run-time errors*).

The size of the profiling table in bytes can be calculated as ASN_PROF_N_COUNTERS * 8.

Refer to Table A3.45 for details on DMA accesses.

A3.10.5 ASN Profiling table flushing

It is admissible for ASN counters in the ASN profiling table to be outdated for an undefined amount of time. This is to enable implementations where event counters are cached in hardware in order to reduce the performance impact on the memory subsystem. Implementations may update the ASN profiling table as an event is triggered, with an IMPLEMENTATION DEFINED frequency, or after a particular event has occurred an IMPLEMENTATION DEFINED number of times.

Software is guaranteed that the ASN profiling table is up to date when receiving a profiling table flush acknowledge message after sending a profiling table flush message to the AMU. Where the AMU has flushed the ASN profiling table of an ASN as a result of the reception of an ASN profiling table flush message, the AMU must send the ASN profiling table flush acknowledge in a timely manner.

When updating a counter in the profiling table, the AMU must do so by writing the 64 bits atomically, such that torn reads cannot take place.

Refer to A4.2.2 *Virtual Function Management Commands and Responses* to see the message formats of the profiling table flush command and response messages.

A3.10.6 Counter overflow notifications

An individual counter overflows when an increment causes a carry-out to be generated, that is when the counter wraps beyond its maximum unsigned value to a smaller value.

When this happens, and the profiling mask is configured to enable notifications for a particular event type, then the AMU sends a counter overflow notification through the exception socket that is part of the management AMI. An overflow condition does not prevent a counter from counting.

Refer to A5.2.2 *Monitoring Exceptions* to see the message format of an ASN counter overflow notification message.

Note

A counter overflow notification can trigger an interrupt by appropriately configuring the digest mask.

Note

Function drivers receive counter overflow notifications through the exception socket in their management AMIs (See A3.4 *Management Interface*).

A3.11 Error detection, reporting and handling

A3.11.1 Overview

This section describes the mechanisms that Revere-AMU provides to notify of any errors that may occur during configuration or message processing.

An error is a condition that prevents the Revere-AMU system from normal operation.

Errors are always reported to a Function driver, called error owner. The error owner is responsible for correcting the condition and notifying the AMU when the condition is corrected.

When an error is detected by the AMU, a specific AMI, a specific Virtual Function, or the Physical Function becomes non-functional until the condition is corrected.

This section focuses on the reporting methods that the Revere-AMU architecture provides. The actual conditions that trigger an error, associated side effects and actions that privileged software is expected to execute are explained in the relevant sections throughout this document.

A3.11.2 Error classification and reporting methods

Revere-AMU errors can occur at configuration time or at run time.

A3.11.2.1 Configuration-time errors

Configuration-time errors can occur when a Function sends a malformed command message or a legal command message with one or more invalid arguments through the Command AMS. Configuration-time errors are always reported through the Response AMS to the same Function that requested the command. The format of these responses and the error codes for each can be found in Chapter A4 *Management Commands and Responses*.

When a configuration-time error is triggered, the AMU:

- Sends an appropriate response message through the Response AMS.
- Stops fetching commands from the Command AMS of the Function in scope.
- Reports this condition by updating AMU_SR and optionally trigger an interrupt, if configured in the ERR_IRQ_CTRL register.

When a configuration-time error occurs, the AMU does not update the read index associated with the command AMS (see A3.4.2 *Management AMI*). Software may check the read index to discover what command triggered the error.

The Function driver must correct the condition and clear AMU_SR. When the condition is corrected, the AMU resumes processing command messages for the Function in scope.

A3.11.2.2 Run-time errors

Run-time errors can occur at any point in time as long as the AMU is enabled (AMU_CR.AMU_EN == 1).

Errors are divided into global, Function and AMI errors. The following table summarises their behaviour:

Error type	Affects	Error Owner	Reporting method
Global	The whole device	PF driver	AMU_SR register
Function	A complete Function	Function driver	AMU_SR register
AMI	A single AMI	Function driver	Exception AMS

When a global run-time error is triggered, the AMU:

- Stops processing messages for the whole device.
- Updates AMU_SR to the appropriate value.

The PF driver must correct the condition and clear AMU_SR. When the condition is corrected, the AMU resumes its normal operations.

When a Function run-time error is triggered, the AMU:

- Stops processing messages for the Function in scope.
- Updates AMU_SR to the appropriate value (see A3.4.3.5 *AMU_SR*).

The Function driver must correct the condition and clear AMU_SR. When the condition is corrected, the AMU resumes processing messages for the Function in scope.

When an AMI run-time error is triggered, the AMU:

- Disables the AMI in scope.
- Sends an exception message through the Exception AMS in the Function in scope.

The Function driver must correct the condition and re-enable the AMI in scope. When the condition is corrected, the AMU resumes processing messages for the AMI in scope.

A3.11.3 Implementation defined errors

Implementations may add IMPLEMENTATION DEFINED global, Function or AMI errors. ID space is reserved for the various reporting methods for this purpose:

- The status that is part of response messages that are sent by the AMU after processing a command message (see Chapter A4 *Management Commands and Responses*).
- The error status that is stored in the AMU_SR register (see A3.4.3.5 *AMU_SR*).
- The exceptions opcode (see Chapter A5 *Exceptions*).

Implementations must ensure that:

- Errors that may affect the behaviour of the whole device are qualified as global errors.
- Errors that may affect the behaviour of a single Function are qualified as Function errors. Such errors must not have any effect on other independent Functions.
- Errors that may affect the behaviour of a single AMI are qualified as AMI errors. Such errors must not have any effect on other independent AMIs, either mapped to the same or to a different Function.
- Errors are reported through the means explained in this section according to their type.

A3.12 QoS controls

Revere-AMU provides mechanisms for software to:

- control the MPAM [7] partition ID for transactions originating from AMU or associated hardware agents (AHAs).
- indicate relative priority of a session (ASN), which gives control over an AMU arbitration scheme, where one is implemented.

A3.12.1 QoS for transactions initiated by the AMU or associated AHAs to Normal memory

If software wants to associate a MPAM's partition ID (PARTID) with AMU or AHA transactions to Normal memory:

- AMU and associated AHAs must use an SMMU that implements the SMMUv3.2 [6] or later architecture for translations.
- AMU and associated AHAs transactions are qualified with PCI Express Requester ID and PASID if stage 1 translation is required.

Using these qualifiers, SMMU adds MPAM information per the SMMU architecture.

Refer to Table A3.45 for details on DMA accesses.

A3.12.2 QoS for transactions initiated by PE to Revere-AMU memory-mapped Registers

If a PE implements MPAM, PE accesses to AMU registers are as memory-mapped IO and use the MPAM information added by the CPU (PE) for the software environment making the access MPAM v1.0. An AMU implementation can use the MPAM information for prioritisation and for partitioning of AMU resources.

A3.12.3 Prioritisation of VFs in a Revere-AMU

Revere-AMU PF may have configurations that set the priority of VFs in using the Revere-AMU's resources. This interface is IMPLEMENTATION DEFINED and not covered by the Revere-AMU system architecture.

A3.12.4 Prioritisation of sessions (ASNs)

ASNs are associated with a priority value. Software can use the priority to determine which session (ASN) to service next. Similarly, the AMU can use the priority to determine the next ASN to service.

A3.13 Virtual Machine Live Migration

A3.13.1 Overview

This section describes the architected mechanisms that Revere-AMU provides to enable the live migration of a virtual machine with one or more Revere-AMU devices attached across two physical machines as well as the requirements that such Revere-AMU devices must fulfil.

Refer to Chapter C2 *VM Live Migration* for a detailed example on how to use these architected capabilities.

A3.13.2 Architecture support

A Revere-AMU device encapsulates internal state, such as registers mapped through the BARs in a Function and AMI, ASN and AMS configuration.

Live migration of Revere-AMU devices is supported through:

- PF management commands aimed at retrieving and restoring all the internal configuration associated to AMIs, ASNs and AMSs.
- PF management commands aimed at quiescing, enabling and disabling Virtual Functions (see A3.4.4.4 *Enable and disable operations*).

Refer to Chapter A4 *Management Commands and Responses* for a complete list of command and response messages.

This enables the PF driver in combination with system software, including the hypervisor, to flush all of the internal AMU buffers associated with a particular Virtual Function to their RX AMSs as well as to retrieve/restore all the state associated with an individual Virtual Function while the migration process is taking place.

A3.13.3 Device requirements

A Revere-AMU device may support live migration, but it is not mandatory.

For a Revere-AMU device to support live migration, the following requirements apply:

- The device must implement AMI type B (see A3.7.5 *AMI-SW types*).
- The Revere-AMU device must have an IMPLEMENTATION DEFINED mechanism to save and restore AHA state.
- The destination host must be identical to the source host. This includes the device/vendor ID as well as the AMU and AHA implementations.
- It must be possible for the destination host to map AMI-SWs and AMI-HWs at the same local IDs in the migrating Virtual Function. Certain restricted mapping schemes may not provide these guarantees.

Support for live migration is also dependent on system software. In particular, system software must provide mechanisms to:

- Notify the PF driver when live migration takes place and exchange state information with the PF driver.
- Enable the PF driver to read/write to the BARs associated with the migrating Virtual Function when such Function is in control of a Virtual Function driver.

A3.14 Power Management

Revere-AMU provides mechanisms for software to control Revere-AMU or associated hardware agents (AHAs) power management.

A Revere-AMU implementation must have support for power management as required by the PCI Express Base Specification [2] from all PCI Express Functions.

Additionally, a Revere-AMU implementation may support IMPLEMENTATION DEFINED power management.

See A3.3 *Revere-AMU and PCI Express*.

A3.14.1 PCI Power States

Revere-AMU Functions must support at the minimum the D0 and D3 PCI power states, as per the PCI Express Base Specification [2]. Revere-AMU implementations may support additional power states such as D1 or D2. See A3.3 *Revere-AMU and PCI Express*.

PCI Power States are defined at the Function level. A Revere-AMU implementation inherits PCI Power States in the following way:

- When an AMI is mapped to an owner Function, it inherits its PCI Power State
- When an AMI is not mapped to any Function, it inherits the PCI Power State of the Physical Function
- All other components of a Revere-AMU implementation such as the AMU and its associated AHAs inherit the PCI Power State of the Physical Function

Table A3.44 summarizes how the PCI Power States are inherited in a Revere-AMU implementation:

Table A3.44: PCI Power States in a Revere-AMU

Component	Mapped	Inherited PCI Power State
AMI	Yes	Function Owner
AMI	No	PF
AMU	-	PF
AHA	-	PF

Note

The PCI Power States have implications on Actions to and from Functions. For example, in $D3_{hot}$:

- Transactions to Functions BAR regions are forbidden
- Transactions from Functions to memory are forbidden, which applies to MSI-X as well

See the PCI Express Base Specification [2].

A3.14.2 Implementation Defined Power States

Revere-AMU implementations may support IMPLEMENTATION DEFINED power states. Those IMPLEMENTATION DEFINED power states must support all the PCI Power States, which are supported. Transitions between the IMPLEMENTATION DEFINED power states must support all the possible transitions between the PCI Power States,

which are supported.

A3.14.3 Wakeup events

The components of a Revere-AMU implementation such as the AMU and its associated AHAs may support wake events. The wake events of a Revere-AMU implementation are attributed to the Physical Function, which means the corresponding PME originate from the Physical Function. See A3.3 *Revere-AMU and PCI Express*.

In the case of an on-chip Revere-AMU implementation, those wake events may be always-on power domain wake events to the Power Controller, as defined in SBSA [3].

An on-chip Revere-AMU Physical Function may be associated with a Root Complex Event Collector, which will receive the Physical Function PMEs. This is outside the scope of this specification.

A3.15 Direct Memory Access

A Revere-AMU may perform accesses to the memory. Its associated hardware agents (AHAs), which have DMA capabilities, may also perform accesses to the memory.

Table A3.45 lists a number of direct memory accesses, which may be performed by a Revere-AMU implementation or its associated AHAs:

Table A3.45: Direct Memory Access summary

Memory access	Base address	Requester ID	PASID	PRI	Error behaviour
MSI-X write[19]	Msg Addr	Function signalling an interrupt	None	No	As per PCIe [2]
Message-signalled interrupt write[20]	IMPLEMENTATION DEFINED	Function signalling an interrupt	None	No	IMPLEMENTATION DEFINED
Management AMI socket ring slot array	MSK_BASE_PTR	Management Interface Function	None	Yes	Report in AMU_SR
Management AMI TYPEB_AMI_SW table[21]	MGT_TYPEB_BASE_PTR	Management Interface Function	None	Yes	Report in AMU_SR
AMI map buffer	AMI_MAP_BASE_PTR	Function executing the F-GET-AMI-MAP command	None	Yes	F-GET-AMI-MAP response STATUS
Profiling table[22]	PROF_TBL_BASE_PTR	Monitoring Function	None	Yes	Report in AMU_SR
AMI-SW TYPEB_AMI_SW table[23]	TYPEB_TBL_BASE_PTR	AMI-SW owning Function	As configured in the AMI-SW	Yes	Report in AMU_SR
AMI-SW AMS ring slot array	RING_BASE_PTR	AMI-SW owning Function	As configured in the AMI-SW	Yes	E-F-AMI-RING-FAULT exception
Message OB buffer table	OB_BUF_TABLE	AMI owning Function	As configured in the AMI	Yes	IMPLEMENTATION DEFINED
Message OB buffer	OB_BUF_PTR	AMI owning Function	As configured in the AMI	Yes	IMPLEMENTATION DEFINED

[19] Only if the Revere-AMU implementation supports interrupts.
[20] For device specific vectors, only if the Revere-AMU implementation supports interrupts.
[21] Only if the Revere-AMU implements AMI type B.
[22] Only when the Revere-AMU implementation supports profiling.
[23] Only if the Revere-AMU implements AMI type B.

When a Revere-AMU implementation does not support PASID, all direct memory accesses performed by the Revere-AMU or its associated AHAs have no PASID.

When a Revere-AMU implementation supports accessing non-pinned memory, the column PRI in Table A3.45 denotes if a direct memory access can trigger a page request.

The address written to by a Revere-AMU Function signalling an MSI-X vector is configured in the Function's MSI-X Table Structure. Each Vector has a corresponding Msg Upper Addr (DWORD 1) and a Msg Addr (DWORD 0), which form a 64-bit address. For more details on MSI-X, refer to the PCI Express Base Specification [2]. For device specific vectors, the behaviour is IMPLEMENTATION DEFINED.

Refer to the following chapters for more details: A3.7.7 *Interrupts*, A3.7 *Accelerator Message Interface for Software*, A3.7.9 *Accessing unpinned memory (SMMU page faults)*, A3.4 *Management Interface*, A3.10 *Profiling*, and A3.12 *QoS controls*.

Chapter A4
Management Commands and Responses

This chapter describes the format and behaviour of commands and responses exchanged through the management AMI.

Following the PCI Express Base Specification [2], Function nomenclature Function N (like F_OWNER) designates the Nth VF if N > 1 or the PF if N = 0.

A4.1 Command and Response opcodes

All command and response messages start with a 16-bit opcode. Opcodes 0x1000-0xFFFF are allocated for IMPLEMENTATION DEFINED command and response messages.

Command messages are always acknowledged, and the response message associated with a command message uses the same opcode as the command message, except if the opcode is invalid in this AMU implementation, in which case the response message uses the following format:

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Invalid received OPCODE
+0x00	[15:0]	OPCODE	Set to 0x0

Responses include a 16-bit status field used to report whether the command was successfully processed. Response status values 0x1000-0xFFFF are allocated for IMPLEMENTATION DEFINED responses.

All PF commands opcodes are deemed invalid when sent through a VF management AMI, and corresponding commands will be forwarded to PF driver or generate an error response (see A3.11.2.1 *Configuration-time errors*) depending on whether trapping is enabled (using PF-F-TRAP-CONFIGURE).

A4.1.1 Management Commands (PF only)

Command Opcode	Command Message
Probing	
TBD	PF-PROBE
TBD	PF-PROBE-AHA
Function Management	
TBD	PF-F-ENABLE
TBD	PF-F-DISABLE
TBD	PF-F-TRAP-CONFIGURE
TBD	PF-F-TRAP-RESP
AMI-SW Management	
TBD	PF-AMI-SW-MAP
TBD	PF-AMI-SW-UNMAP
TBD	PF-AMI-SW-CONFIGURE
TBD	PF-AMI-SW-SAVE
TBD	PF-AMI-SW-RESET
AMI-HW Management	
TBD	PF-AMI-HW-MAP
TBD	PF-AMI-HW-UNMAP
TBD	PF-AMI-HW-CONFIGURE
TBD	PF-AMI-HW-SAVE
TBD	PF-AMI-HW-RESET
ASN Management	
TBD	PF-ASN-CREATE
TBD	PF-ASN-DESTROY
AMS Management	

Command Opcode	Command Message
TBD	PF-AMS-RING-CONFIGURE
TBD	PF-AMS-RING-SAVE

A4.1.2 Management Commands (All Functions)

Command Opcode	Command Message
Probing	
TBD	F-PROBE
TBD	F-GET-AMI-MAP
Function Management	
TBD	F-PROF-TABLE-FLUSH[1]
AMI-SW Management	
TBD	F-AMI-SW-CONFIGURE
TBD	F-AMI-SW-SAVE
TBD	F-AMI-SW-RESET
TBD	F-AMI-SW-ENABLE
TBD	F-AMI-SW-DISABLE
AMI-HW Management	
TBD	F-AMI-HW-CONFIGURE
TBD	F-AMI-HW-SAVE
TBD	F-AMI-HW-RESET
TBD	F-AMI-HW-ENABLE
TBD	F-AMI-HW-DISABLE
AMS Management	
TBD	F-AMS-RING-CONFIGURE
TBD	F-AMS-RING-SAVE
TBD	F-AMS-PROF-CONFIGURE[2]
TBD	F-AMS-TRACE-CONFIGURE[3]

A4.2 Command and Response formats

A4.2.1 Probing Commands and Responses

A4.2.1.1 PF-PROBE

A4.2.1.1.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode

[1] Only when the Revere-AMU implementation supports profiling.
[2] Only when the Revere-AMU implementation supports profiling.
[3] Only if the Revere-AMU implementation supports tracing.

Used by the PF driver to retrieve a set of architected capabilities not discoverable through the management registers (see A3.4.4.1.1 *Constants fixed by implementation*).

This command is always processed by the AMU.

A4.2.1.1.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: Command successful
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	MAX_AMI_SW	Total number of AMI-SWs implemented
+0x08	[31:21]	- Reserved -	Reserved, RES0
+0x08	[20:0]	MAX_ASN	Maximum number of ASNs that can be created
+0x0C	[31:16]	- Reserved -	Reserved, RES0
+0x0C	[15:0]	NUM_AHA	Number of AHAs in the system

A4.2.1.2 PF-PROBE-AHA

A4.2.1.2.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AHA_ID	ID of the AHA

Used by the PF driver to retrieve a set of architected capabilities from each AHA.

This command is always processed by the AMU.

A4.2.1.2.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: capabilities retrieved successfully 1: Invalid AHA_ID
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:0]	- Reserved -	Reserved, RES0
+0x08	[63:0]	AHA_IIDR	Information about the implementation and implementer of the AHA

A4.2.1.3 F-PROBE

A4.2.1.3.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode

Used by Function drivers to retrieve a set of architected capabilities not discoverable through the management registers (see A3.4.4.1.1 *Constants fixed by implementation*).

This command is always processed by the AMU.

A4.2.1.3.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: Command successful
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:0]	ASN_PROF_N_COUNTERS	Number of ASN profiling counters implemented
+0x08	[31:22]	- Reserved -	Reserved, RES0
+0x08	[21]	PASID_SUPPORTED	PASID is supported by the implementation
+0x08	[20:16]	PASID_WIDTH	PASID width supported by the implementation
+0x08	[15:0]	AMI_PER_F_M1	Maximum number (minus one) of AMI supported per Function AMI_PER_F = (AMI_PER_F_M1 + 1)

When the Revere-AMU does not support profiling, ASN_PROF_N_COUNTERS is reserved, RES0.

A4.2.1.4 F-GET-AMI-MAP

A4.2.1.4.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:0]	- Reserved -	Reserved, RES0
+0x08	[63:12]	AMI_MAP_BASE	Address of a buffer used to return the AMI map
+0x08	[11:0]	- Reserved -	Reserved, RES0

Used by Function drivers to retrieve the AMI map (see A3.4.4.3 *Mapping AMIs to Functions*).

AMI_MAP_BASE_PTR is the 64-bit virtual address of the AMI map buffer which is aligned on 4KB.

AMI_MAP_BASE_PTR[63:12] are provided by AMI_MAP_BASE and AMI_MAP_BASE_PTR[11:0] are treated as 0s.

The AMI map buffer must be at least 16KB in size and will be accessed by the AMU using the Requester ID of the requesting Function with no PASID.

Refer to Table A3.45 for details on DMA accesses.

This command is always processed by the AMU.

A4.2.1.4.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: AMI map retrieved successfully 1: Fault when accessing AMI map buffer
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.2 Virtual Function Management Commands and Responses

A4.2.2.1 PF-F-ENABLE

A4.2.2.1.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	F_OWNER	Function to be enabled. Cannot be 0

Used by the PF driver to enable a VF (see A3.4.4.4 *Enable and disable operations*).

If Function F_OWNER is already enabled this command has no effect.

A4.2.2.1.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: Function enabled 1: F_OWNER is invalid
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.2.2 PF-F-DISABLE

A4.2.2.2.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	F_OWNER	Function to be disabled. Cannot be 0

Used by the PF driver to disable a VF. Disabling a VF has the effect of deactivating all AMIs assigned to the VF, including the management AMI (see A3.4.4.4 *Enable and disable operations*).

If Function F_OWNER is already disabled this command has no effect.

A4.2.2.2.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: Function disabled 1: F_OWNER is invalid
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:1]	- Reserved -	Reserved, RES0
+0x04	[0]	TRAP_CONFIG	Trapping status in VF

A4.2.2.3 PF-F-TRAP-CONFIGURE

A4.2.2.3.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:17]	- Reserved -	Reserved, RES0
+0x04	[16]	ENABLE	0: disable trapping 1: enable trapping
+0x04	[15:0]	F_OWNER	Function where trapping is enabled or disabled Cannot be 0

Used by the PF driver to enable trapping of all the Function management commands targeting any unmapped logical AMI or with an unrecognized opcode (IMPDEF or invalid) into its own exception session (see A3.4.4.2 *AMU management command trapping*).

After Function level reset AMI trapping is disabled.

This command is illegal when F_OWNER is enabled.

A4.2.2.3.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: Success 1: F_OWNER is invalid 2: F_OWNER is enabled
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.2.4 PF-F-TRAP-RESP

A4.2.2.4.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	F_TARGET	destination VF number
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	STATUS	Status Code
+0x04	[15:0]	RESP_OPCODE	Response Opcode
+0x08		PAYLOAD	Use-case specific payload

Used by the PF driver to send a response to the VF driver (see A3.4.4.2 *AMU management command trapping*) for a trapped command. The semantics of the payload may be IMPLEMENTATION DEFINED. The AMU will strip the first 32b of the command before forwarding and will rely on a non-zero STATUS field to trigger an error.

This command is processed by the AMU when the receiving Function is enabled.

A4.2.2.4.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: Message Delivered to VF 1: Message could not be delivered
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.2.5 F-PROF-TABLE-FLUSH

A4.2.2.5.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode

Used by a Function driver to request that all ASN profiling tables associated with sessions (ASN) that the Function owns are flushed to main memory (see A3.10 *Profiling*).

The associated response is received once all the tables have been flushed.

When the Revere-AMU does not support profiling, this command is invalid.

This command is always processed by the AMU.

A4.2.2.5.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: Success 1: Failure
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.3 AMI-SW Management Commands and Responses

A4.2.3.1 PF-AMI-SW-MAP

A4.2.3.1.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode

Offset	Bits	Name	Description
+0x04	[31:17]	- Reserved -	Reserved, RES0
+0x04	[16]	ANY_PHY_AMI_SW	1: Map any physical AMI-SW 0: Map PHY_AMI_SW_ID
+0x04	[15:0]	PHY_AMI_SW_ID	Physical AMI-SW ID to be mapped
+0x08	[31:16]	F_OWNER	Function to map the AMI-SW to
+0x08	[15:0]	AMI_SW_ID	ID to assign to the AMI-SW within F_OWNER

Used by the PF driver to map a physical AMI-SW to a Function (see A3.4.4.3 *Mapping AMIs to Functions*).

When ANY_PHY_AMI_SW == 1, the AMU attempts to map any physical AMI-SW in the system and PHY_AMI_SW_ID is ignored. When ANY_PHY_AMI_SW == 0, the AMU attempts to map PHY_AMI_SW_ID.

AMI_SW_ID must be available within F_OWNER for the command to succeed.

This command is always processed by the AMU.

A4.2.3.1.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW mapped successfully 1: F_OWNER is invalid 2: PHY_AMI_SW_ID already mapped 3: AMI_SW_ID already assigned in F_OWNER
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	PHY_AMI_SW_ID	Physical AMI-SW ID mapped if ANY_PHY_AMI_SW == 1

A4.2.3.2 PF-AMI-SW-UNMAP

A4.2.3.2.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	PHY_AMI_SW_ID	Physical AMI-SW ID to be unmapped

Used by the PF driver to unmap an AMI-SW from a Function.

When it succeeds, the AMI-SW is also quiesced (see A3.4.4.4.3 *AMI Quiescing*).

If PHY_AMI_SW_ID is not mapped, the message has no effect.

This command is always processed by the AMU.

A4.2.3.2.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code

Offset	Bits	Name	Description
+0x00	[15:0]	OPCODE	0: AMI-SW unmapped successfully Command Opcode

A4.2.3.3 PF-AMI-SW-CONFIGURE

A4.2.3.3.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	PHY_AMI_SW_ID	Physical AMI-SW ID to be configured
+0x08	[31:21]	- Reserved -	Reserved, RES0
+0x08	[20]	PASID_ENABLED	1: Enable PASID 0: Disable PASID
+0x08	[19:0]	PASID	PASID value
+0x0C	[31]	TX_IRQ_ENABLE	1: Enable TX Interrupt Generation 0: Disable TX Interrupt Generation
+0x0C	[30:11]	- Reserved -	Reserved, RES0
+0x0C	[10:0]	TX_VECTOR	Value to use for TX_VECTOR
+0x10	[31]	RX_IRQ_ENABLE	1: Enable RX Interrupt Generation 0: Disable RX Interrupt Generation
+0x10	[30:11]	- Reserved -	Reserved, RES0
+0x10	[10:0]	RX_VECTOR	Value to use for RX_VECTOR
+0x14	[31:0]	- Reserved -	Reserved, RES0
+0x18	[63:12]	TYPEB_TBL_BASE	Bits [63:12] of the address of TYPEB_AMI_SW table
+0x18	[11:1]	- Reserved -	Reserved, RES0
+0x18	[0]	ENABLE	1: Enable AMI-SW 0: Disable AMI-SW

Used by the PF driver to configure an AMI-SW in a Function (see A3.4.4.1.2 *AMI-SW Configuration*).

This command is illegal when the AMI-SW is mapped.

When the Revere-AMU does not implement PASID capabilities, PASID_ENABLED and PASID are reserved, RES0.

When the Revere-AMU implementation does not support interrupts, RX_IRQ_ENABLE, TX_IRQ_ENABLE, RX_VECTOR and TX_VECTOR are reserved, RES0.

When the Revere-AMU implementation supports interrupts, changing the value of the RX_IRQ_ENABLE or TX_IRQ_ENABLE parameters can generate an interrupt under circumstances described in A3.7.7 *Interrupts*.

TYPEB_TBL_BASE_PTR is the 64-bit virtual address of the TYPEB_AMI_SW table which is aligned on 4KB.

TYPEB_TBL_BASE_PTR[63:12] are provided by TYPEB_TBL_BASE and TYPEB_TBL_BASE_PTR[11:0] are treated as 0s.

When Revere-AMU does not implement AMI Type B, TYPEB_TBL_BASE is reserved, RES0.

A4.2.3.3.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW configured successfully 1: AMI-SW is mapped
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.3.4 PF-AMI-SW-SAVE

A4.2.3.4.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	PHY_AMI_SW_ID	Physical AMI-SW ID to be saved

Used by the PF driver to retrieve the state of an AMI-SW (see A3.4.4.1.2 *AMI-SW Configuration*).

This command is illegal when the AMI is currently mapped onto an enabled Function.

A4.2.3.4.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW saved successfully 1: Invalid PHY_AMI_SW_ID 2: Owning Function is enabled
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:21]	- Reserved -	Reserved, RES0
+0x04	[20]	PASID_ENABLED	1: PASID is enabled 0: PASID is disabled
+0x04	[19:0]	PASID	PASID value
+0x08	[31]	TX_IRQ_ENABLE	1: Enable TX Interrupt Generation 0: Disable TX Interrupt Generation
+0x08	[30:11]	- Reserved -	Reserved, RES0
+0x08	[10:0]	TX_VECTOR	TX_VECTOR value
+0x0C	[31]	RX_IRQ_ENABLE	1: Enable RX Interrupt Generation 0: Disable RX Interrupt Generation
+0x0C	[30:11]	- Reserved -	Reserved, RES0
+0x0C	[10:0]	RX_VECTOR	RX_VECTOR value
+0x10	[63:12]	TYPEB_TBL_BASE	Bits [63:12] of the address of TYPEB_AMI_SW table
+0x10	[11:2]	- Reserved -	Reserved, RES0
+0x10	[1]	MAPPED	1: AMI-SW is mapped 0: AMI-SW is unmapped
+0x10	[0]	ENABLE	1: AMI-SW is enabled 0: AMI-SW is disabled
+0x18	[31:16]	F_OWNER	Function onto which AMI-SW is mapped (if mapped)
+0x18	[15:0]	AMI_SW_ID	AMI-SW ID within the function (if mapped)

When the Revere-AMU does not implement PASID capabilities, PASID_ENABLED and PASID are reserved, RES0.

When the Revere-AMU implementation does not support interrupts, RX_IRQ_ENABLE, TX_IRQ_ENABLE, RX_VECTOR and TX_VECTOR are reserved, RES0.

A4.2.3.5 PF-AMI-SW-RESET

A4.2.3.5.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	PHY_AMI_SW_ID	Physical AMI-SW ID to be reset

Used by the PF driver to reset an AMI-SW (see A3.4.4.5 *Reset domains*).

This command is illegal when the AMI-SW is mapped.

A4.2.3.5.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW reset successfully 1: AMI-SW is mapped
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.3.6 F-AMI-SW-CONFIGURE

A4.2.3.6.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_SW_ID	AMI-SW within the Function to be configured
+0x08	[31:21]	- Reserved -	Reserved, RES0
+0x08	[20]	PASID_ENABLED	1: Enable PASID 0: Disable PASID
+0x08	[19:0]	PASID	PASID value
+0x0C	[31]	TX_IRQ_ENABLE	1: Enable TX Interrupt Generation 0: Disable TX Interrupt Generation
+0x0C	[30:11]	- Reserved -	Reserved, RES0
+0x0C	[10:0]	TX_VECTOR	Value to use for TX_VECTOR
+0x10	[31]	RX_IRQ_ENABLE	1: Enable RX Interrupt Generation 0: Disable RX Interrupt Generation
+0x10	[30:11]	- Reserved -	Reserved, RES0
+0x10	[10:0]	RX_VECTOR	Value to use for RX_VECTOR
+0x14	[31:0]	- Reserved -	Reserved, RES0

Offset	Bits	Name	Description
+0x18	[63:12]	TYPEB_TBL_BASE	Bits [63:12] of the address of TYPEB_AMI_SW table
+0x18	[11:0]	- Reserved -	Reserved, RES0

Used by a Function driver to configure an AMI-SW (see A3.4.4.1.2 *AMI-SW Configuration*).

This command is illegal when the AMI-SW is enabled.

When the Revere-AMU does not implement PASID capabilities, PASID_ENABLED and PASID are reserved, RES0.

When the Revere-AMU implementation does not support interrupts, RX_IRQ_ENABLE, TX_IRQ_ENABLE, RX_VECTOR and TX_VECTOR are reserved, RES0.

When the Revere-AMU implementation supports interrupts, changing the value of the RX_IRQ_ENABLE or TX_IRQ_ENABLE parameters can generate an interrupt under circumstances described in A3.7.7 *Interrupts*.

TYPEB_TBL_BASE_PTR is the 64-bit virtual address of the TYPEB_AMI_SW table which is aligned on 4KB.

TYPEB_TBL_BASE_PTR[63:12] are provided by TYPEB_TBL_BASE and TYPEB_TBL_BASE_PTR[11:0] are treated as 0s.

When Revere-AMU does not implement AMI Type B, TYPEB_TBL_BASE is reserved, RES0.

A4.2.3.6.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW configured successfully 1: AMI_SW_ID is not mapped in Function 2: AMI-SW is not disabled
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.3.7 F-AMI-SW-SAVE

A4.2.3.7.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_SW_ID	AMI-SW within the Function to be saved

Used by a Function driver to retrieve the state of an AMI-SW (see A3.4.4.1.2 *AMI-SW Configuration*).

This command is always processed by the AMU.

A4.2.3.7.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW saved successfully

Offset	Bits	Name	Description
			1: invalid AMI_SW_ID
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:21]	- Reserved -	Reserved, RES0
+0x04	[20]	PASID_ENABLED	1: PASID is enabled 0: PASID is disabled
+0x04	[19:0]	PASID	PASID value
+0x08	[31]	TX_IRQ_ENABLE	1: Enable TX Interrupt Generation 0: Disable TX Interrupt Generation
+0x08	[30:11]	- Reserved -	Reserved, RES0
+0x08	[10:0]	TX_VECTOR	TX_VECTOR value
+0x0C	[31]	RX_IRQ_ENABLE	1: Enable RX Interrupt Generation 0: Disable RX Interrupt Generation
+0x0C	[30:11]	- Reserved -	Reserved, RES0
+0x0C	[10:0]	RX_VECTOR	RX_VECTOR value
+0x10	[63:12]	TYPEB_TBL_BASE	Bits [63:12] of the address of TYPEB_AMI_SW table
+0x10	[11:0]	- Reserved -	Reserved, RES0

When the Revere-AMU does not implement PASID capabilities, PASID_ENABLED and PASID are reserved, RES0.

When the Revere-AMU implementation does not support interrupts, RX_IRQ_ENABLE, TX_IRQ_ENABLE, RX_VECTOR and TX_VECTOR are reserved, RES0.

A4.2.3.8 F-AMI-SW-RESET

A4.2.3.8.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_SW_ID	AMI-SW within the Function to be reset

Used by a Function driver to reset an AMI-SW (see A3.4.4.5 *Reset domains*).

This command is always processed by the AMU.

A4.2.3.8.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW reset successfully 1: Invalid AMI_SW_ID
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.3.9 F-AMI-SW-ENABLE

A4.2.3.9.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_SW_ID	AMI-SW within the Function to be enabled

Used by a Function driver to enable an AMI-SW (see A3.4.4.4 *Enable and disable operations*).

This command is always processed by the AMU.

If the AMI-SW is already enabled this command has no effect.

A4.2.3.9.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW enabled successfully 1: AMI-SW is not mapped in Function
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.3.10 F-AMI-SW-DISABLE

A4.2.3.10.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_SW_ID	AMI-SW within the Function to be enabled

Used by a Function driver to disable an AMI-SW (see A3.4.4.4 *Enable and disable operations*).

This command is always processed by the AMU.

If the AMI-SW is already disabled this command has no effect.

A4.2.3.10.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-SW disabled successfully 1: AMI-SW is not mapped in Function
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.4 AMI-HW Management Commands and Responses

A4.2.4.1 PF-AMI-HW-MAP

A4.2.4.1.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:17]	- Reserved -	Reserved, RES0
+0x04	[16:0]	AHA_ID	ID of the AHA that contains the physical AMI-HW
+0x08	[31:17]	- Reserved -	Reserved, RES0
+0x08	[16]	ANY_PHY_AMI_HW	1: Map any physical AMI-HW 0: Map PHY_AMI_HW_ID
+0x08	[15:0]	PHY_AMI_HW_ID	Physical AMI-HW ID within AHA_ID to be mapped
+0x0C	[31:16]	F_OWNER	Function to map the AMI-HW to
+0x0C	[15:0]	AMI_HW_ID	ID to assign to the AMI-HW within F_OWNER

Used by the PF driver to map a physical AMI-HW within an AHA to a Function (see A3.4.4.3 *Mapping AMIs to Functions*).

When ANY_PHY_AMI_HW == 1, the AMU attempts to map any available physical AMI-HW in the AHA and PHY_AMI_HW_ID is ignored. When ANY_PHY_AMI_HW == 0, the AMU attempts to map PHY_AMI_HW_ID.

AMI_HW_ID must be available within F_OWNER for the command to succeed.

This command is illegal when the AMI-SW is mapped.

A4.2.4.1.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW mapped successfully 1: F_OWNER is invalid 2: PHY_AMI_HW_ID in AHA_ID already mapped 3: AMI_HW_ID already assigned in F_OWNER
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	PHY_AMI_HW_ID	Physical AMI-HW ID assigned

A4.2.4.2 PF-AMI-HW-UNMAP

A4.2.4.2.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	AHA_ID	ID of the AHA that contains the physical AMI-HW
+0x04	[15:0]	PHY_AMI_HW_ID	Physical AMI-HW ID within AHA_ID to be unmapped

Used by the PF driver to unmap an AMI-HW from a Function.

When it succeeds, the AMI-HW is also quiesced (see A3.4.4.4.3 *AMI Quiescing*).

This command is always processed by the AMU.

If AMI_HW is not mapped, the message has no effect.

A4.2.4.2.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: AMI-HW unmapped successfully
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.4.3 PF-AMI-HW-CONFIGURE

A4.2.4.3.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	AHA_ID	ID of the AHA that contains the physical AMI-HW
+0x04	[15:0]	PHY_AMI_HW_ID	Physical AMI-HW ID within AHA_ID to be configured
+0x08	[31:26]	- Reserved -	Reserved, RES0
+0x08	[25]	ENABLE	1: Enable AMI-HW 0: Disable AMI-HW
+0x08	[24:21]	LOG2_HW_CRED_SIZE	Credit size to use for this AMI-HW in Doublewords as $\log_2$(HW_CRED_SIZE)
+0x08	[20]	PASID_ENABLED	1: Enable PASID 0: Disable PASID
+0x08	[19:0]	PASID	PASID value

Used by the PF driver to configure an AMI-HW in a Function (see A3.4.4.1.3 *AMI-HW Configuration*).

This command is illegal when the AMI-HW is mapped.

LOG2_HW_CRED_SIZE value must not be greater than AMU_IDR.MIN_LOG2_MSG_LENGTH.

When the Revere-AMU does not implement PASID capabilities, PASID_ENABLED and PASID are reserved, RES0.

A4.2.4.3.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW configured successfully 1: Invalid LOG2_HW_CRED_SIZE value 2: PHY_AMI_HW_ID within AHA_ID is mapped
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.4.4 PF-AMI-HW-SAVE

A4.2.4.4.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	AHA_ID	ID of the AHA that contains the physical AMI-HW
+0x04	[15:0]	PHY_AMI_HW_ID	Physical AMI-HW ID within AHA_ID to be saved

Used by the PF driver to retrieve the state of an AMI-HW (see A3.4.4.1.3 *AMI-HW Configuration*).

This command is illegal when the AMI is currently mapped onto an enabled Function.

A4.2.4.4.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW saved successfully 1: Invalid PHY_AMI_HW_ID 2: Owning Function is enabled
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:26]	- Reserved -	Reserved, RES0
+0x04	[25]	ENABLE	1: AMI-HW is enabled 0: AMI-HW is enabled
+0x04	[24:21]	LOG2_HW_CRED_SIZE	Credit size being used for this AMI-HW in Doublewords as $\log_2$(HW_CRED_SIZE)
+0x04	[20]	PASID_ENABLED	1: PASID is enabled 0: PASID is disabled
+0x04	[19:0]	PASID	PASID value

When the Revere-AMU does not implement PASID capabilities, PASID_ENABLED and PASID are reserved, RES0.

A4.2.4.5 PF-AMI-HW-RESET

A4.2.4.5.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	AHA_ID	ID of the AHA that contains the physical AMI-HW
+0x04	[15:0]	PHY_AMI_HW_ID	Physical AMI-HW ID within AHA_ID to be reset

Used by the PF driver to reset an AMI-HW (see A3.4.4.5 *Reset domains*).

This command is illegal when the AMI-HW is mapped.

A4.2.4.5.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW reset successfully 1: AMI-HW is mapped
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.4.6 F-AMI-HW-CONFIGURE

A4.2.4.6.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_HW_ID	AMI-HW within the Function to be configured
+0x08	[31:21]	- Reserved -	Reserved, RES0
+0x08	[20]	PASID_ENABLED	1: Enable PASID 0: Disable PASID
+0x08	[19:0]	PASID	PASID value

Used by a Function driver to configure an AMI-HW (see A3.4.4.1.3 *AMI-HW Configuration*).

This command is illegal when the AMI-HW is enabled.

When the Revere-AMU does not implement PASID capabilities, PASID_ENABLED and PASID are reserved, RES0.

A4.2.4.6.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW configured successfully 1: AMI-HW is not mapped in Function 2: AMI-HW is not disabled
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.4.7 F-AMI-HW-SAVE

A4.2.4.7.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_HW_ID	AMI-HW within the Function to be saved

Used by a Function driver to retrieve the state of an AMI-HW (see A3.4.4.1.3 *AMI-HW Configuration*).

This command is always processed by the AMU.

A4.2.4.7.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW saved successfully 1: Invalid AMI_HW_ID
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:21]	- Reserved -	Reserved, RES0
+0x04	[20]	PASID_ENABLED	1: PASID is enabled 0: PASID is disabled
+0x04	[19:0]	PASID	PASID value

When the Revere-AMU does not implement PASID capabilities, PASID_ENABLED and PASID are reserved, RES0.

A4.2.4.8 F-AMI-HW-RESET

A4.2.4.8.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_HW_ID	AMI-HW within the Function to be reset

Used by a Function driver to reset an AMI-HW (see A3.4.4.5 *Reset domains*).

This command is always processed by the AMU.

A4.2.4.8.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW reset successfully 1: Invalid AMI_HW_ID
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.4.9 F-AMI-HW-ENABLE

A4.2.4.9.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0

Offset	Bits	Name	Description
+0x04	[15:0]	AMI_HW_ID	AMI-HW within the Function to be enabled

Used by a Function driver to enable an AMI-HW (see A3.4.4.4 *Enable and disable operations*).

This command is always processed by the AMU.

If the AMI-HW is already enabled this command has no effect.

A4.2.4.9.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW enabled successfully 1: AMI-HW is not mapped in Function
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.4.10 F-AMI-HW-DISABLE

A4.2.4.10.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_HW_ID	AMI-HW within the Function to be disabled

Used by a Function driver to disable an AMI-HW (see A3.4.4.4 *Enable and disable operations*).

This command is always processed by the AMU.

If the AMI-HW is already disabled this command has no effect.

A4.2.4.10.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMI-HW disabled successfully 1: AMI-HW is not mapped in Function
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.5 ASN Management Commands and Responses

A4.2.5.1 PF-ASN-CREATE

A4.2.5.1.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:22]	- Reserved -	Reserved, RES0
+0x04	[21:6]	MON_OWNER	Monitoring owner
+0x04	[5]	SRC_AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x04	[4]	DST_AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x04	[3:0]	PRIO	Priority to use for this ASN
+0x08	[31:16]	SRC_AHA_ID	Reserved if SRC_AMI_TYPE=0 ID of the AHA containing the sending physical AMI-HW if SRC_AMI_TYPE=1
+0x08	[15:0]	SRC_PHY_AMI_ID	Physical AMI-SW ID for the sending AMI if SRC_AMI_TYPE=0 Physical AMI-HW ID within AHA_ID for the sending AMI if SRC_AMI_TYPE=1
+0x0C	[31:16]	DST_AHA_ID	Reserved if DST_AMI_TYPE=0 ID of the AHA containing the receiving physical AMI-HW if DST_AMI_TYPE=1
+0x0C	[15:0]	DST_PHY_AMI_ID	Physical AMI-SW ID for the receiving AMI if DST_AMI_TYPE=0 Physical AMI-HW ID within AHA_ID for the receiving AMI if DST_AMI_TYPE=1
+0x10	[31:13]	- Reserved -	Reserved, RES0
+0x10	[12]	STASH_ENABLE	Enable stashing for this ASN
+0x10	[11:6]	SRC_AMS	Sending AMS within SRC_AMI
+0x10	[5:0]	DST_AMS	Receiving AMS within DST_AMI
+0x14	[31:28]	- Reserved -	Reserved, RES0
+0x14	[27:0]	ASN_ID	Physical identifier for this ASN
+0x18	[63:0]	STASH_CTL	Stashing controls to use for this ASN
+0x20	[63:0]	PROF_MASK	Profiling mask
+0x28	[63:3]	PROF_TBL_BASE	Bits[63:3] of the 64-bit Profiling table base pointer PROF_TBL_BASE_PTR
+0x28	[2:0]	- Reserved -	Reserved, RES0
+0x30	[31:16]	TRACE_CTL	Tracing controls to use for this ASN
+0x30	[15:0]	PROF_CTL	Profiling controls to use for this ASN
+0x34	[31:0]	STASH_DEST	Physical destination for stashing
+0x38	[31:14]	- Reserved -	Reserved, RES0
+0x38	[13:10]	LOG2_MSG_LENGTH	Message length (in Doublewords) as $\log_2$(MSG_LENGTH)
+0x38	[9:3]	MF_OB_BUF_NUM	Number of references to out-of-band buffers
+0x38	[2:0]	MFO	Message format to use for this ASN

Used by the PF driver to create a new ASN (see A3.4.4.1.4 *ASN Configuration*).

When the Revere-AMU implementation does not support profiling, PROF_MASK, PROF_TBL_BASE and PROF_CTL are reserved, RES0.

When the Revere-AMU implementation does not support tracing, TRACE_CTL is reserved, RES0.

LOG2_MSG_LENGTH shall be contained in the range allowed by the implementation A3.7.2 *Ring buffers*.

Both sending and receiving AMSes must not be already connected to an ASN for this command to succeed.

A4.2.5.1.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: ASN created successfully 1: Requested ASN ID already in use 2: LOG2_MSG_LENGTH out of range. 3: AMS already connected to an ASN
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.5.2 PF-ASN-DESTROY

A4.2.5.2.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:28]	- Reserved -	Reserved, RES0
+0x04	[27:0]	ASN_ID	Physical identifier for ASN to be destroyed

Used by the PF driver to destroy an existing ASN (see A3.4.4.1.4 *ASN Configuration*).

In addition to destroying the session, the AMU:

- Quiesces the ASN.
- Returns the state associated with the ASN.

A4.2.5.2.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	STATUS	Status Code 0: ASN destroyed successfully 1: ASN does not exist
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:22]	- Reserved -	Reserved, RES0
+0x04	[21:6]	MON_OWNER	Monitoring owner
+0x04	[5]	SRC_AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x04	[4]	DST_AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x04	[3:0]	PRIO	Priority to use for this ASN
+0x08	[31:16]	SRC_AHA_ID	Reserved if SRC_AMI_TYPE=0 ID of the AHA containing the sending physical AMI-HW if SRC_AMI_TYPE=1
+0x08	[15:0]	SRC_PHY_AMI_ID	Physical AMI-SW ID for the sending AMI if SRC_AMI_TYPE=0 Physical AMI-HW ID within AHA_ID for the sending AMI if SRC_AMI_TYPE=1
+0x0C	[31:16]	DST_AHA_ID	Reserved if DST_AMI_TYPE=0 ID of the AHA containing the receiving physical AMI-HW if DST_AMI_TYPE=1
+0x0C	[15:0]	DST_PHY_AMI_ID	Physical AMI-SW ID for the receiving AMI if DST_AMI_TYPE=0 Physical AMI-HW ID within AHA_ID for the receiving AMI if DST_AMI_TYPE=1

Offset	Bits	Name	Description
+0x10	[31:13]	- Reserved -	Reserved, RES0
+0x10	[12]	STASH_ENABLE	Enable stashing for this ASN
+0x10	[11:6]	SRC_AMS	Sending AMS within sending AMI
+0x10	[5:0]	DST_AMS	Receiving AMS within receiving AMI
+0x14	[31:16]	TRACE_CTL	Tracing controls configured in this ASN
+0x14	[15:0]	PROF_CTL	Profiling controls configured in this ASN
+0x18	[63:0]	STASH_CTL	Stashing controls configured in this ASN
+0x20	[63:0]	PROF_MASK	Profiling mask
+0x28	[63:3]	PROF_TBL_BASE	Bits[63:3] of the 64-bit Profiling table base pointer PROF_TBL_BASE_PTR
+0x28	[2:0]	- Reserved -	Reserved, RES0
+0x30	[31:0]	STASH_DEST	Physical destination for stashing
+0x34	[31:14]	- Reserved -	Reserved, RES0
+0x34	[13:10]	LOG2_MSG_LENGTH	Message length (in Doublewords) as $\log_2$(MSG_LENGTH)
+0x34	[9:3]	MF_OB_BUF_NUM	Number of references to out-of-band buffers
+0x34	[2:0]	MFO	Message format to use for this ASN

When the Revere-AMU implementation does not support tracing, TRACE_CTL is reserved, RES0.

When the Revere-AMU implementation does not support profiling, PROF_CTL, PROF_MASK and PROF_TBL_BASE are reserved, RES0.

A4.2.6 AMS Management Commands and Responses

A4.2.6.1 PF-AMS-RING-CONFIGURE

A4.2.6.1.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	PHY_AMI_SW_ID	Physical AMI-SW ID containing AMS to be configured
+0x08	[31:7]	- Reserved -	Reserved, RES0
+0x08	[6]	AMS_TYPE	0: RX AMS 1: TX AMS
+0x08	[5:0]	AMS_ID	AMS within AMI_SW to be configured
+0x0C	[31:10]	- Reserved -	Reserved, RES0
+0x0C	[9:6]	THRESHOLD	Controls the change of TX_DIGEST or RX_DIGEST bits
+0x0C	[5:1]	LOG2_SIZE	Size of the ring in slots as $\log_2$(SIZE)
+0x0C	[0]	RX_MODE	0: Back-pressure 1: Overwriting
+0x10	[63:3]	RING_BASE	Bits [63:3] of the 64-bit virtual address of the slot array (RING_BASE_PTR)
+0x10	[2:0]	- Reserved -	Reserved, RES0

Used by the PF driver to configure an AMS ring buffer in an AMI-SW (see A3.4.4.1.5 *AMS Configuration*).

This command is processed by the AMU only when the AMI is unmapped.

A4.2.6.1.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMS ring configured successfully 1: AMI-SW is mapped 2: LOG2_SIZE out of range
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.6.2 PF-AMS-RING-SAVE

A4.2.6.2.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	PHY_AMI_SW_ID	Physical AMI-SW ID containing AMS to be saved
+0x08	[31:7]	- Reserved -	Reserved, RES0
+0x08	[6]	AMS_TYPE	0: RX AMS 1: TX AMS
+0x08	[5:0]	AMS_ID	AMS within AMI_SW to be saved

Used by the PF driver to retrieve the state of an AMS ring buffer in an AMI-SW (see A3.4.4.1.5 *AMS Configuration*).

This command is illegal when the AMI is currently mapped onto an enabled Function.

A4.2.6.2.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMS ring saved successfully 1: Owning Function is enabled
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:10]	- Reserved -	Reserved, RES0
+0x04	[9:6]	THRESHOLD	Controls the change of TX_DIGEST or RX_DIGEST bits
+0x04	[5:1]	LOG2_SIZE	Size of the ring in slots as $\log_2$(SIZE)
+0x04	[0]	RX_MODE	0: Back-pressure 1: Overwriting
+0x08	[63:3]	RING_BASE	Bits [63:3] of the 64-bit virtual address of the slot array (RING_BASE_PTR)
+0x08	[2:0]	- Reserved -	Reserved, RES0

A4.2.6.3 F-AMS-RING-CONFIGURE

A4.2.6.3.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	AMI_SW_ID	AMI_SW within the Function to be configured
+0x08	[31:7]	- Reserved -	Reserved, RES0
+0x08	[6]	AMS_TYPE	0: RX AMS 1: TX AMS
+0x08	[5:0]	AMS_ID	AMS within AMI_SW to be configured
+0x0C	[31:10]	- Reserved -	Reserved, RES0
+0x0C	[9:6]	THRESHOLD	Controls the change of TX_DIGEST or RX_DIGEST bits
+0x0C	[5:1]	LOG2_SIZE	Size of the ring in slots as $\log_2$(SIZE)
+0x0C	[0]	RX_MODE	0: Back-pressure 1: Overwriting
+0x10	[63:3]	RING_BASE	Bits [63:3] of the 64-bit virtual address of the slot array (RING_BASE_PTR)
+0x10	[2:0]	- Reserved -	Reserved, RES0

Used by a Function driver to configure an AMS ring buffer in an AMI-SW (see A3.4.4.1.5 *AMS Configuration*).

This command is illegal when the AMI-SW that contains the AMS is enabled.

A4.2.6.3.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMS ring configured successfully 1: AMI-SW is not mapped in Function 2: LOG2_SIZE out of range 3: AMI-SW is not disabled
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.6.4 F-AMS-RING-SAVE

A4.2.6.4.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:23]	- Reserved -	Reserved, RES0
+0x04	[22]	AMS_TYPE	0: RX AMS 1: TX AMS
+0x04	[21:16]	AMS_ID	AMS within AMI_SW to be saved
+0x04	[15:0]	AMI_SW_ID	AMI_SW within the Function to be saved

Used by a Function driver to retrieve the state of an AMS ring buffer in an AMI-SW (see A3.4.4.1.5 *AMS Configuration*).

This command is always processed by the AMU.

A4.2.6.4.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: AMS ring saved successfully 1: Invalid AMI_SW_ID 2: Invalid AMS_ID
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:11]	- Reserved -	Reserved, RES0
+0x04	[10:7]	THRESHOLD	Controls the change of TX_DIGEST or RX_DIGEST bits
+0x04	[6:2]	LOG2_SIZE	Size of the ring in slots as $\log_2$(SIZE)
+0x04	[1]	RX_MODE	0: Back-pressure 1: Overwriting
+0x04	[0]	- Reserved -	Reserved, RES0
+0x08	[63:3]	RING_BASE	Bits [63:3] of the 64-bit virtual address of the slot array (RING_BASE_PTR)
+0x08	[2:0]	- Reserved -	Reserved, RES0

A4.2.6.5 F-AMS-PROF-CONFIGURE

A4.2.6.5.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:24]	- Reserved -	Reserved, RES0
+0x04	[23]	AMS_TYPE	0: RX AMS 1: TX AMS
+0x04	[22:17]	AMS_ID	AMS within AMI to be configured
+0x04	[16]	AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x04	[15:0]	AMI_ID	AMI within F_OWNER to be configured
+0x08	[63:0]	PROF_MASK	Profiling mask
+0x10	[63:3]	PROF_TBL_BASE	Bits[63:3] of the 64-bit Profiling table base pointer PROF_TBL_BASE_PTR
+0x10	[2:0]	- Reserved -	Reserved, RES0
+0x18	[31:16]	- Reserved -	Reserved, RES0
+0x18	[15:0]	PROF_CTL	Profiling controls configured in the associated ASN

Used by a Function driver to configure profiling for the ASN connected to an AMS in an AMI (see A3.10 *Profiling*).

This command is only valid when the implementation supports profiling.

This command can only succeed when the sending Function is the monitoring owner of the underlying ASN.

This command is always processed by the AMU.

A4.2.6.5.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code

Offset	Bits	Name	Description
			0: Profiling configured successfully 1: Invalid AMI_ID 2: Invalid AMS_ID 3: Function is not the monitoring owner
+0x00	[15:0]	OPCODE	Command Opcode

A4.2.6.6 F-AMS-TRACE-CONFIGURE

A4.2.6.6.1 Command

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Command Opcode
+0x04	[31:24]	- Reserved -	Reserved, RES0
+0x04	[23]	AMS_TYPE	0: RX AMS 1: TX AMS
+0x04	[22:17]	AMS_ID	AMS within AMI to be configured
+0x04	[16]	AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x04	[15:0]	AMI_ID	AMI within F_OWNER to be configured
+0x08	[31:16]	- Reserved -	Reserved, RES0
+0x08	[15:0]	TRACE_CTL	Tracing controls configured in the associated ASN

Used by a Function driver to configure tracing for the ASN connected to an AMS in an AMI (see A3.9 *Tracing*).

This command is only valid when the implementation supports tracing.

This command can only succeed when the sending Function is the monitoring owner of the underlying ASN.

This command is always processed by the AMU.

A4.2.6.6.2 Response

Offset	Bits	Name	Description
+0x00	[31:16]	Status	Status Code 0: Tracing configured successfully 1: Invalid AMI_ID 2: Invalid AMS_ID 3: Function is not the monitoring owner
+0x00	[15:0]	OPCODE	Command Opcode

Chapter A5
Exceptions

This chapter describes the format and behaviour of exceptions generated and sent by the AMU through the management AMI.

A5.1 Exception opcodes

All exception messages start with a 16-bit opcode. Opcodes 0x1000-0xFFFF are allocated for IMPLEMENTATION DEFINED exceptions.

PF Only	Exception Opcode	Exception Message
Yes	TBD	E-PF-VF-RESET
No	TBD	E-F-COUNTER-OVERFLOW
No	TBD	E-F-AMI-RING-FAULT
No	TBD	E-F-AMI-RING-INVALID-VECTOR
Yes	TBD	E-PF-TRAPPED-CMD

A5.2 Exception formats

A5.2.1 Virtual Function Management Exceptions

A5.2.1.1 E-PF-VF-RESET

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Exception Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	SOURCE	Source VF

Triggered when a VF has received a Function level reset (see A3.4.4.5 *Reset domains*).

The PF driver may use this notification to clear any state associated with the Function.

A5.2.2 Monitoring Exceptions

A5.2.2.1 E-F-COUNTER-OVERFLOW

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Exception Opcode
+0x04	[31:16]	F_OWNER	Function that owns the AMI_SW
+0x04	[15:0]	AMI_ID	AMI within F_OWNER
+0x08	[31:8]	- Reserved -	Reserved, RES0
+0x08	[7]	AMS_TYPE	0: RX AMS 1: TX AMS
+0x08	[6]	AMI_TYPE	0: AMI-SW 1: AMI-HW
+0x08	[5:0]	AMS_ID	AMS within AMI_SW to be configured
+0x0C	[31:16]	- Reserved -	Reserved, RES0
+0x0C	[15:0]	COUNTER_ID	Counter that overflowed

Triggered when an ASN performance counter associated with a session (ASN) that is owned by the Function overflows and the counter is configured to trigger a notification (see A3.10 *Profiling*).

A5.2.3 Error Exceptions

A5.2.3.1 E-F-AMI-RING-FAULT

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Exception Opcode
+0x04	[31:22]	- Reserved -	Reserved, RES0
+0x04	[21:16]	AMS_ID	AMS within AMI_SW that triggered the fault
+0x04	[15:0]	AMI_ID	AMI within F_OWNER

Offset	Bits	Name	Description
+0x08	[31:0]	- Reserved -	Reserved, RES0

Triggered when the AMU faults when accessing an AMS ring slot array (see A3.7.2 *Ring buffers*).

Refer to Table A3.45 for details on DMA accesses.

A5.2.3.2 E-F-AMI-RING-INVALID-VECTOR

Offset	Bits	Name	Description
+0x00	[31:16]	- Reserved -	Reserved, RES0
+0x00	[15:0]	OPCODE	Exception Opcode
+0x04	[31:17]	- Reserved -	Reserved, RES0
+0x04	[16]	VECTOR_TYPE	0: RX_VECTOR 1: TX_VECTOR
+0x04	[15:0]	AMI_ID	AMI within F_OWNER
+0x08	[31:0]	- Reserved -	Reserved, RES0

Triggered when an AMI-SW TX_VECTOR or RX_VECTOR is invalid (see A3.7.7.3 *Configuration*).

A5.2.4 Trapping Exceptions

A5.2.4.1 E-PF-TRAPPED-CMD

Offset	Bits	Name	Description
+0x00	[31:16]	F_SOURCE_ID	Source VF number
+0x00	[15:0]	OPCODE	Exception Opcode
+0x04	[31:16]	- Reserved -	Reserved, RES0
+0x04	[15:0]	TRAPPED_OPCODE	Trapped command Opcode
+0x08		TRAPPED_CMD	Trapped command body

Triggered when the AMU traps a command (see A3.4.4.2 *AMU management command trapping*).

Part B
Pin-level interface for hardware agents

Chapter B1
Scope

This section defines the optional pin-level interface for hardware accelerators, which is invisible to software but enables reuse of AHAs and a standard IP implementation of the AMU.

This section describes the following procedures:

- Managing a connection with an hardware agent (AHA)
- Configuring AMI-HW and AMS in an AHA
- Sending and receiving Revere-AMU messages over an ASN

Including the following aspects:

- Packet format
- Flow control scheme
- Transport layer specification (AXI4STREAM based)
- DMA by an AHA
- Stream and substream ID for use with an SMMU.
- Stashing.
- Compliance with PCI Express Function requirements (for example: PCI Express Bus Master Enable bit (BME) and Transactions Pending (TP) register).
- Programming interfaces needed to manage messaging to AHAs, in particular configuring ASNs between AHAs

---

Note

Power control and integration with SMMU will be addressed in a later release Slave access to AHA from a PF using this pin level interface will also be addressed in a later release.

Chapter B2
AMU AHA Interface protocol

B2.1 Introduction

This section introduces the AMU AHA Interface (AAI) protocol and describes the components of an AAI-compliant implementation.

The AMU AHA Interface describes the interface between the AMU and the AHAs and is optional. This interface supports independent development of an AMU and an AHA.

The AAI protocol may be used by implementations of the Revere-AMU to connect AMU and AHAs. Arm recommends an AHA implementation uses this interface.

A communication channel that providing a packet interface, based on the AMBA AXI-4 Stream Transport, is required for each direction:

- From the AMU to the AHA
- From the AHA to the AMU

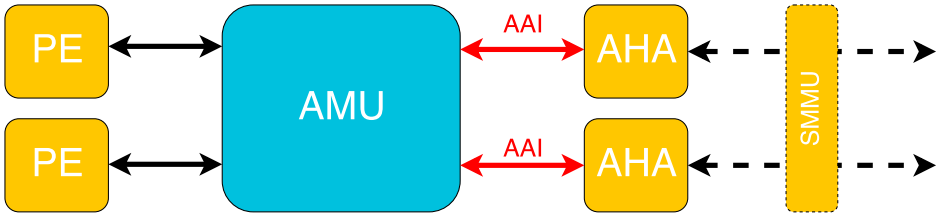

Figure B2.1: AAI overview

This document describes the version 1.0 of this protocol.

B2.1.1 Terminology

The AAI protocol is a point-to-point protocol. Each channel consists of a link, an AAI master, and an AAI slave. The AAI master is the AMU, and the AAI slave is the AHA. Downstream communication is defined as communication from AMU to AHA Upstream communication is defined as communication from AHA to AMU

The AAI is composed by the following layers:

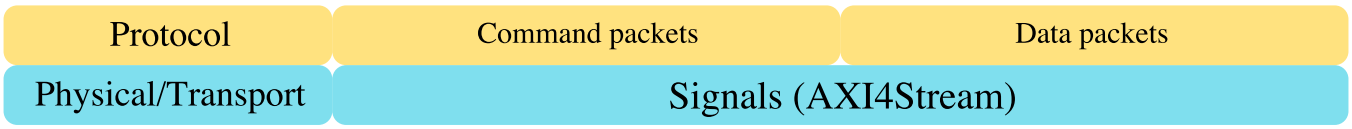

Figure B2.2: AAI layers

Data packets are a special kind of packets with a different flow control mechanism (credits), and are encapsulating the Revere-AMU messages as seen by software.

B2.2 AAI Protocol overview

This chapter provides an overview of the AAI protocol.

B2.2.1 AAI protocol

The AAI Protocol supports four types of packets:

- Command packets for control actions.
- Response command packets that acknowledge command packets, used for token return when necessary.
- Data packets sent over Revere-AMU sessions (ASN).
- Response data packets acknowledging those data packets, used for credits return.

Each command packet has an equivalent handshake response packet that acknowledges the command, which is sent for each received command. Each data packet has an equivalent handshake response data packet that returns credits.

Unlike command packets, a response data packet is not guaranteed to be sent for each data packet received. They are sent when one or more messages are consumed in the RX AMS.

B2.2.2 Packet groups

AAI protocol packets are grouped below according to function. The following table shows the AAI packet groups:

Table B2.1: Packet groups of the AAI protocol

Packet group	source	Protocol function
Connection and disconnection	AMU	Establishes, terminates or resets the connection between the AMU and the AHA
AMI management	AMU	Enables, disables or resets an AMI-HW
Session management	AMU	Connects, disconnects, or updates an AMS-HW
Messaging	AMU/AHA	sends/acks Revere-AMU messages over AAI protocol
Credits management	AMU/AHA	manages ASN credits exchanges
DMA management	AMU	propagates PCI Express BME and Transactions Pending

Note

Packets to convey register accesses to AHAs and trace data from AHAs are under consideration for an upcoming release.

B2.2.3 Managing AAI connections

B2.2.3.1 Channel states

The five possible states of the AAI channel are:

B2.2.3.1.1 DISCONNECTED

The channel is disconnected. The AAI master and the AAI slave cannot issue packets until the connection has been established.

B2.2.3.1.2 REQ_CONNECT

The AAI master has issued a Connect Request. The AAI slave must provide an appropriate handshaking response to either establish or reject the connection.

B2.2.3.1.3 CONNECTED

The channel is connected. The AAI master and AAI slave can issue packets as permitted by the protocol rules.

B2.2.3.1.4 REQ_DISCONNECT

The AAI master has issued a Disconnect Request. The AAI slave issues a Disconnect Acknowledge in response.

B2.2.3.1.5 REQ_RESET

The AAI master has issued an AHA Reset Request. The AAI slave issues a reset Acknowledge in response.

B2.2.3.2 Handshaking

On powerup, or after an AHA reset, the channel between the AAI master and the AAI slave is initially in the DISCONNECTED state. The following figure shows how the channel state of the channel changes in response to connect and disconnect/reset packets.

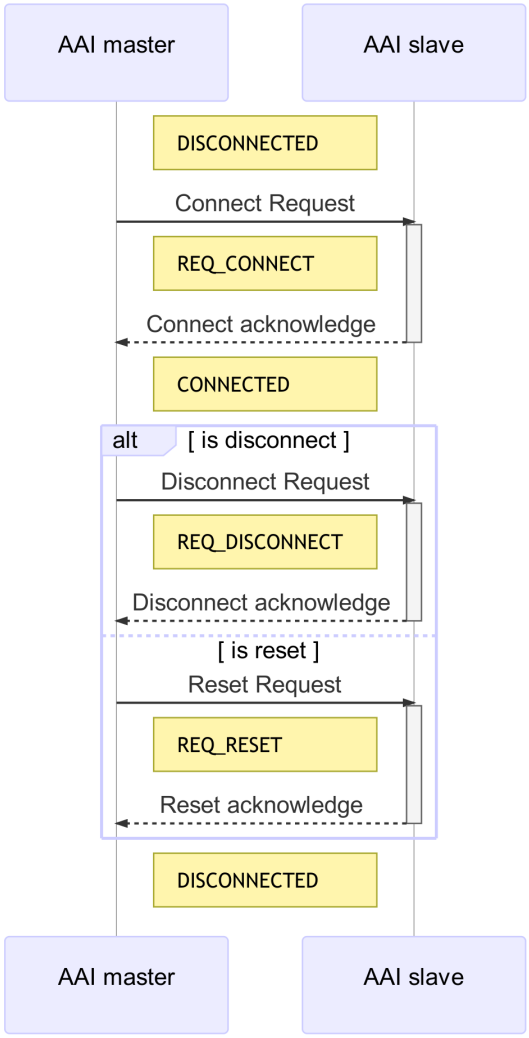

Figure B2.3: AMU PF Interface Sequence diagram.

The following table describes the connection or disconnection packets that are permitted in each channel state:

Table B2.2: Permitted packets in each AAI channel state

Channel state	AAI master permitted packets	AAI slave permitted packets
DISCONNECTED	Connect Request only	None
REQ_CONNECT	None	Connect acknowledge
CONNECTED	Any, subject to the protocol rules	Any, subject to the protocol rules
REQ_DISCONNECT	ACK only	Any, subject to the protocol rules
REQ_RESET	None	Reset acknowledge

B2.3 AAI Protocol Packets

AAI packets are a fixed length and are a whole number of bytes in size. The transport medium must preserve the correct number of bytes for each packet.

There is a one-to-one mapping between a protocol packet and a transport packet (they are the same).

The declared size of an AAI packet is always a multiple of the implemented datapath width used for the stream transfer.

Where the number of bytes required by a packet is less than the overall packet size, the unused bytes are marked as reserved and filled with the value zero.

B2.3.1 AAI packets listing

The four least significant bits of every packet are used to encode the packet-type (MST_PKT_TYPE or SLV_PKT_TYPE field, see below).

All other command and response IDs beyond those defined in this specification are reserved.

If any endpoint receives a reserved ID, this constitutes a protocol error. The packet-type encodings are defined independently for packets originating from:

- AAI masters.
- AAI slaves.

Note

Some encoding combinations are common to both encoding spaces.

The following table shows the master-initiated packets of the AAI protocol:

Table B2.3: AAI protocol master-initiated packets

Packet group	Packet	MST_PKT_TYPE field encoding
Messaging	MSG_SEND	0x1
Messaging	MSG_SEND_ACK	0x2
Credits management	CRED_REQ	0x3
Credits management	CRED_REQ_ACK	0x4
Credits management	CRED_RETURN	0x5
Credits management	CRED_RETURN_ACK	0x6
Credits management	CRED_RECALL	0x7
Credits management	CRED_RECALL_ACK	0x8
Credits management	CRED_SEND	0x9
Credits management	CRED_SEND_ACK	0xA

Packet group	Packet	MST_PKT_TYPE field encoding
Connection and disconnection	AHA_CONDIS_REQ	0xB
Connection and disconnection	AHA_RESET_REQ	0xC
AMI management	AMI_ENADIS_REQ	0xD
AMI management	AMI_RESET_REQ	0xE
Session management	RX_AMS_CONDIS_REQ	0xF
Session management	TX_AMS_CONDIS_REQ	0x10
DMA management	DMA_BME_REQ	0x11
DMA management	DMA_TRANS_PEND_REQ	0x12

The following table shows the slave-initiated packets of the AAI protocol:

Table B2.4: AAI protocol slave-initiated packets

Packet group	Packet	SLV_PKT_TYPE field encoding
Messaging	MSG_SEND	0x1
Messaging	MSG_SEND_ACK	0x2
Credits management	CRED_REQ	0x3
Credits management	CRED_REQ_ACK	0x4
Credits management	CRED_RETURN	0x5
Credits management	CRED_RETURN_ACK	0x6
Credits management	CRED_RECALL	0x7
Credits management	CRED_RECALL_ACK	0x8
Credits management	CRED_SEND	0x9
Credits management	CRED_SEND_ACK	0xA
Connection and disconnection	AHA_CONDIS_ACK	0xB
Connection and disconnection	AHA_RESET_ACK	0xC
AMI management	AMI_ENADIS_ACK	0xD
AMI management	AMI_RESET_ACK	0xE
Session management	RX_AMS_CONDIS_ACK	0xF
Session management	TX_AMS_CONDIS_ACK	0x10

Packet group	Packet	SLV_PKT_TYPE field encoding
DMA management	DMA_BME_ACK	0x11
DMA management	DMA_TRANS_PEND_ACK	0x12

B2.3.2 Reserved fields

Reserved fields in packets are described as Should Be Zero (SBZ). The recipient of a packet with reserved fields must ignore these fields. This specification recommends that the sender drive a reserved field to 0.

B2.3.3 IMPDEF fields

Some packet fields are defined as being IMPLEMENTATION DEFINED. These fields can be used by implementations for any defined purpose. These fields are treated as Reserved by components that do not require them.

B2.3.4 Software generation of protocol errors

Software programming must never be permitted to cause a hardware protocol error because this might result in the AHA or AMU becoming non-operational.

B2.3.5 Hardware generation of packet errors

If packets are sent that do not correspond to the architected format described in this specification, then the architecture makes no guarantees about the behaviour of the AMU. In such cases, the AMU can become non-operational in many ways, and it is CONSTRAINED UNPREDICTABLE whether an implementation:

- Ignores the packet (tokens or credits can be lost in this case)
- Sends a notification through the PF exception AMS. See A3.4 *Management Interface*.
- Signals a fault with an IMPLEMENTATION DEFINED mechanism

Physical damage to the system cannot be precluded in some implementations.

B2.3.6 Flow control

The AAI protocol uses tokens and credits to provide flow control.

B2.3.6.1 Command packets

Tokens are used to manage the number of command packets, of different types, that can be outstanding at a point in time. The AAI protocol uses the following types of command token:

- AMI tokens: Used in AMI management requests to limit the number of outstanding AMI management requests.
- ASN tokens: Used in ASN management requests to limit the number of outstanding ASN management requests.
- DMA tokens: Used in DMA management requests to limit the number of outstanding DMA management requests.

Tokens are grouped this way by source event. There is no dependency between packets belonging to same token group to avoid deadlocks.

Note

Connection and disconnection packets do not require tokens as master can only issue one such packet per AHA at any given time. Credit management packets do not require tokens as each ASN requires one and only one such packet.

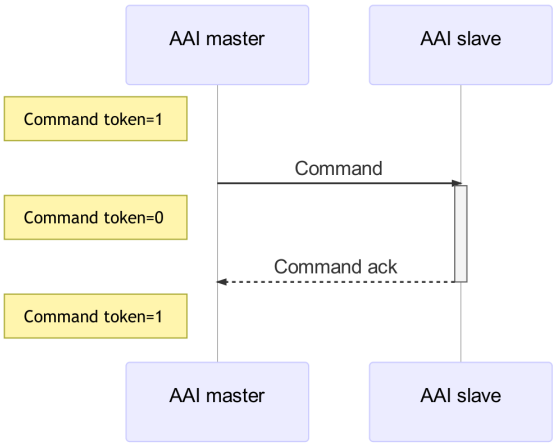

Figure B2.4: Token based flow control for command packets.

B2.3.6.2 Data packets

Credits are used to manage the number of data packets that can be sent by a socket (AMS) through a Revere session (ASN) at a given time. The AAI protocol uses the following type of credits:

- ASN credits: Used to credit Reveres ASNs (data packets).

One ASN credit is used to credit a Revere-AMU message size (including descriptor) of HW_CRED_SIZE bytes for MFO1. For other MFOs, one ASN credit is used to credit exactly one Revere-AMU message.

The packetization overhead is not taken into account to compute the number of required credits before sending a data packet for MFO1.

Only the full Revere-AMU message size (including descriptor) is accounted for.

During AMI configuration, the AHA is free to reject any unsupported value for HW_CRED_SIZE (by acknowledging the request with STATE=0) but once accepted this value shall be used as credit size in bytes for all sessions belonging to the given AMI.

Credits are returned using the data ack packet (MSG_SEND_ACK). A TX-AMS can send data packets as long as it owns enough credits to do so and the credits spent are not greater than the maximum allowed by the RX side. The number of credits actually consumed is the greater value between the credits consumed and the minimum credits consumption allowed by the RX side if ASN has MFO1.

Credits can return in any number of data ack packets, as long as this number is no greater than the number of data packets initially sent.

Example with HW_CRED_SIZE=64:

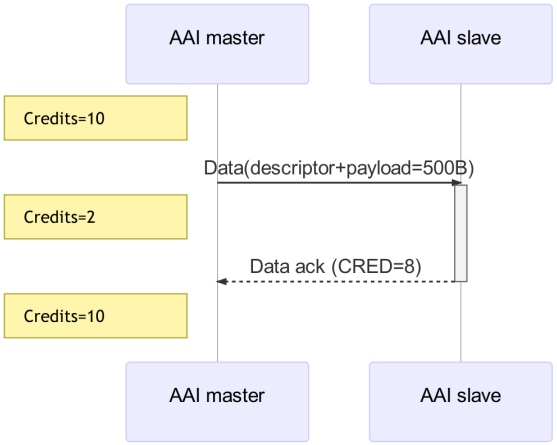

Figure B2.5: Credit based flow control for data packets.

Revere-AMU pin level interface architecture mandates that for AMI-HW to AMI-HW communications with MFO1, the HW_CRED_SIZE of the RX AMI shall be equal to the HW_CRED_SIZE of the TX AMI.

Note

Only the data packets have a priority hint (PRIO field). This field can be used:

- By AHAs to prioritise ASNs (select which ASN to service first)
- By a NoC or interconnect inside AMU to prioritise traffic.
- By the AMU to prioritise ASNs (select which ASN to service first)

B2.4 AAI packets details

B2.4.1 Connection and disconnection packet group

This section describes the connection and disconnection packet group.

B2.4.1.1 AHA_CONDIS_REQ

B2.4.1.1.1 Description

This packet is used to initiate a connection or disconnection handshake. It is a connection state change request.

B2.4.1.1.2 Source

AMU

B2.4.1.1.3 Usage constraints

The AAI master can only send a connect request when:

- The channel is in the DISCONNECTED state.

The AAI master can only send a disconnect request when:

- The channel is in the CONNECTED state.
- There are no outstanding AMU initiated token based request.

B2.4.1.1.4 Flow control result

None

B2.4.1.1.5 Fields description

The AHA_CONDIS_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
VERSION		This field identifies the requested protocol version. An AAI master can request any protocol version it supports.
STATE		This bit identifies the new channel state requested. 0:disconnect request. 1:connect request
TOK_AMI_REQ		The meaning of this field depends on the value of the STATE field. **When STATE = 0:** This field indicates the number of AMI management tokens returned The number of AMI management tokens returned is equal to the value of this field plus one. This field must be the value of TOK_AMI_GNT that was received in the AHA_CONDIS_ACK packet that acknowledged the connection of the channel **When STATE = 1:** This field indicates the number of AMI management tokens requested. The number of AMI management tokens requested is equal to the value of this field plus one.

Field name	bits	description
TOK_AMS_REQ		The meaning of this field depends on the value of the STATE field. **When STATE = 0:** This field indicates the number of session management tokens returned The number of session management tokens returned is equal to the value of this field plus one. This field must be the value of TOK_AMS_GNT that was received in the AHA_CONDIS_ACK packet that acknowledged the connection of the channel **When STATE = 1:** This field indicates the number of session management tokens requested. The number of session management tokens requested is equal to the value of this field plus one.
TOK_DMA_REQ		The meaning of this field depends on the value of the STATE field. **When STATE = 0:** This field indicates the number of DMA management tokens returned The number of DMA management tokens returned is equal to the value of this field plus one. This field must be the value of TOK_DMA_GNT that was received in the AHA_CONDIS_ACK packet that acknowledged the connection of the channel **When STATE = 1:** This field indicates the number of DMA management tokens requested. The number of DMA management tokens requested is equal to the value of this field plus one.

B2.4.1.2 AHA_CONDIS_ACK

B2.4.1.2.1 Description

The AHA_CONDIS_ACK packet is used to accept or deny a request as part of the connection or disconnection handshake process. It is a connection state change acknowledgement.

B2.4.1.2.2 Source

AHA

B2.4.1.2.3 Usage constraints

The AAI master must have previously issued an unacknowledged AHA_CONDIS_REQ packet.

B2.4.1.2.4 Flow control result

None

B2.4.1.2.5 Fields description

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
VERSION		The protocol version that is granted by the AAI slave The value of this field must not be greater than the value of the VERSION field in the AHA_CONDIS_REQ packet received.
STATE		This bit identifies the new state. The possible values of this bit are:

Field name	bits	description
		0:DISCONNECTED. 1:CONNECTED When the value of STATE in the unacknowledged AHA_CONDIS_REQ packet is 0, the value of this bit must be 0. When the value of STATE in the unacknowledged AHA_CONDIS_REQ packet is 1, this field can be 0 or 1. 0 normally indicates a serious system configuration failure.
TOK_AMI_GNT		This field indicates the number of pre-allocated tokens for AMI management requests that have been granted. The number of AMI management tokens granted is equal to the value of this field plus one. The value of this field must not be greater than the value of the TOK_AMI_REQ field in the AHA_CONDIS_REQ packet. When the value of STATE is 0, this field is ignored.
TOK_AMS_GNT		This field indicates the number of pre-allocated tokens for session management requests that have been granted. The number of session management tokens granted is equal to the value of this field plus one. The value of this field must not be greater than the value of the TOK_AMS_REQ field in the AHA_CONDIS_REQ packet. When the value of STATE is 0, this field is ignored.
TOK_DMA_GNT		This field indicates the number of pre-allocated tokens for DMA management requests that have been granted. The number of DMA management tokens granted is equal to the value of this field plus one. The value of this field must not be greater than the value of the TOK_DMA_REQ field in the AHA_CONDIS_REQ packet. When the value of STATE is 0, this field is ignored.
ERROR		When the value of STATE in the unacknowledged AHA_CONDIS_REQ packet is 1, and STATE value is 0, this field gives the reason why an error happened. Otherwise ignored

B2.4.1.3 AHA_RESET_REQ

B2.4.1.3.1 Description

This packet is used to initiate a full AHA reset handshake. It is also a connection state change request.

B2.4.1.3.2 Source

AMU

B2.4.1.3.3 Usage constraints

The AAI master can only send a reset request when the channel is in the CONNECTED state.

B2.4.1.3.4 Flow control result

None

B2.4.1.3.5 Fields description

The AHA_RESET_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings

B2.4.1.4 AHA_RESET_ACK

B2.4.1.4.1 Description

This packet is used to acknowledge a reset request.

B2.4.1.4.2 Source

AHA

B2.4.1.4.3 Usage constraints

The AAI master must have previously issued a reset request that has not yet generated either a response or an error packet.

B2.4.1.4.4 Flow control result

None

B2.4.1.4.5 Fields description

The AHA_RESET_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings

B2.4.2 AMI management packet group

This section describes the AMI management packet group.

B2.4.2.1 AMI_ENADIS_REQ

B2.4.2.1.1 Description

This packet is used to enable an AMI in an AHA. It is an AMI enable request.

B2.4.2.1.2 Source

AMU

B2.4.2.1.3 Usage constraints

The AAI master can only send an AMI enable request when:

- The channel is in the CONNECTED state.
- The AAI master has at least one AMI management token.

B2.4.2.1.4 Flow control result

The AAI master sends an AMI management token to the AAI slave.

B2.4.2.1.5 Fields description

The AMI_ENADIS_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
STATE		This bit identifies the new AMI state requested. 0:AMI disable request. 1:AMI enable request
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.
SID		This field indicates the StreamID that has to be used for all DMA accesses associated with this AMI This field is ignored if the AHA does not have a DMA, or if STATE value is 0
SSID		This field indicates the SubstreamID that has to be used for all DMA accesses This field is ignored if the AHA does not have a DMA, if the SMMU does not implement stage 1 translation or if STATE value is 0
LOG2_HW_CRED_SIZE		This field indicates the credit size in Doublewords as $\log_2$(HW_CRED_SIZE) for Revere messages exchanges over ASNs belonging to the targeted AMI This field is ignored if STATE value is 0

Refer to Table A3.45 for details on DMA accesses.

B2.4.2.2 AMI_ENADIS_ACK

B2.4.2.2.1 Description

This packet is used to acknowledge an AMI enable request.

B2.4.2.2.2 Source

AHA

B2.4.2.2.3 Usage constraints

The AAI master must have previously issued an AMI enable request that has not yet generated either a response or an error packet.

B2.4.2.2.4 Flow control result

The AAI slave returns an AMI management token to the AAI master.

B2.4.2.2.5 Fields description

The AMI_ENADIS_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
STATE		This bit identifies the new state. The possible values of this bit are: 0:AMI DISABLED. 1:AMI ENABLED When the value of STATE in the unacknowledged AMI_ENADIS_REQ packet is 0, the value of this bit must be 0. When the value of STATE in the unacknowledged AMI_ENADIS_REQ packet is 1, this field can be 0 or 1. 0 normally indicates a serious system configuration failure.
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.
ERROR		When the value of STATE in the unacknowledged AMI_ENADIS_REQ packet is 1, and STATE value is 0, this field gives the error code. Otherwise ignored.

B2.4.2.3 AMI_RESET_REQ

B2.4.2.3.1 Description

This packet is used to initiate a full AMI reset handshake.

B2.4.2.3.2 Source

AMU

B2.4.2.3.3 Usage constraints

The AAI master can only send a reset request when the channel is in the CONNECTED state and the AMI is in ENABLED state.

B2.4.2.3.4 Flow control result

The AAI master sends an AMI management token to the AAI slave.

B2.4.2.3.5 Fields description

The AMI_RESET_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.

B2.4.2.4 AMI_RESET_ACK

B2.4.2.4.1 Description

This packet is used to acknowledge an AMI reset request.

B2.4.2.4.2 Source

AHA

B2.4.2.4.3 Usage constraints

The AAI master must have previously issued an AMI reset request that has not yet generated either a response or an error packet.

B2.4.2.4.4 Flow control result

The AAI slave returns an AMI management token to the AAI master.

B2.4.2.4.5 Fields description

The AMI_RESET_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.

B2.4.3 Session management packet group

This section describes the Session management packet group.

B2.4.3.1 RX_AMS_CONDIS_REQ

B2.4.3.1.1 Description

This packet is used to connect an ASN to an AMS in an AHA. It is an RX AMS connection request.

B2.4.3.1.2 Source

AMU

B2.4.3.1.3 Usage constraints

The AAI master can only send an RX AMS connection request when:

- The channel is in the CONNECTED state.
- The AMI is in the ENABLED state.
- The AAI master has at least one session management token.

B2.4.3.1.4 Flow control result

The AAI master sends a session management token to the AAI slave.

B2.4.3.1.5 Fields description

The RX_AMS_CONDIS_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
STATE		This bit identifies the new AMS state requested. 0:AMS disconnect request. 1:AMS connect request
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.
AMS_ID		This field indicates the RX AMS_ID targeted by this request.
MFO		This field indicates the message format requested for the ASN being connected
LOG2_MSG_LENGTH		Message length (in Doublewords) as $\log_2$(MSG_LENGTH)

B2.4.3.2 RX_AMS_CONDIS_ACK

B2.4.3.2.1 Description

This packet is used to acknowledge an RX AMS connection request.

B2.4.3.2.2 Source

AHA

B2.4.3.2.3 Usage constraints

The AAI master must have previously issued an AMS connection request that has not yet generated either a response or an error packet.

B2.4.3.2.4 Flow control result

The AAI slave returns a session management token to the AAI master.

B2.4.3.2.5 Fields description

The RX_AMS_CONDIS_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
STATE		This bit identifies the new state. The possible values of this bit are: 0:AMS disconnected. 1:AMS connected When the value of STATE in the unacknowledged RX_AMS_CONDIS_REQ packet is 0, the value of this bit must be 0. When the value of STATE in the unacknowledged RX_AMS_CONDIS_REQ packet is 1, this field can be 0 or 1. 0 normally indicates a serious system configuration failure.

Field name	bits	description
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID initiating this request.
AMS_ID		This field indicates the RX AMS_ID initiating this request.
CRED_GNT		This field indicates the number of credits that have been allocated to the session connected to the newly enabled RX AMS. The number of credits granted is equal to the value of this field plus one. When the value of STATE is 0, this field is ignored.
MIN_CRED_USE		This field indicates the minimum amount of credits consumed by a message sent to this AMS
MAX_CRED_USE		This field indicates the maximum amount of credits a message received by this AMS can use
ERROR		When the value of STATE in the unacknowledged RX_AMS_CONDIS_REQ packet is 1, and STATE value is 0, this field gives the error code. Otherwise ignored.

B2.4.3.3 TX_AMS_CONDIS_REQ

B2.4.3.3.1 Description

This packet is used to connect an AMS in an AHA. It is a TX AMS connection request.

B2.4.3.3.2 Source

AMU

B2.4.3.3.3 Usage constraints

The AAI master can only send a TX AMS connection request when:

- The channel is in the CONNECTED state.
- The AAI master has at least one session management token.

B2.4.3.3.4 Flow control result

The AAI master sends a session management token to the AAI slave.

B2.4.3.3.5 Fields description

The TX_AMS_CONDIS_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
STATE		This bit identifies the new AMS state requested. 0:AMS disconnect request. 1:AMS connect request
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.

Field name	bits	description
AMS_ID		This field indicates the TX AMS_ID targeted by this request.
CRED_GNT		This field indicates the number of credits that have been allocated to the session connected to the newly enabled TX AMS. The number of credits granted is equal to the value of this field plus one. When the value of STATE is 0, this field is ignored.
LOG2_HW_CRED_SIZE		This field indicates the credit size in Doublewords as $\log_2$(HW_CRED_SIZE) used by the ASN being connected. When the value of STATE is 0, this field is ignored.
MIN_CRED_USE		This field indicates the minimum amount of credits consumed by a message going through this AMS. When the value of STATE is 0, this field is ignored.
MAX_CRED_USE		This field indicates the maximum amount of credits a message going through this AMS can use. When the value of STATE is 0, this field is ignored.
PRIO		This field is used to set the priority for data packets sent through this AMS. When the value of STATE is 0, this field is ignored.
STASH_DEST		This field indicates the stashing destination to use. Ignored if stashing is not supported. When the value of STATE is 0, this field is ignored.
MFO		This field indicates the message format requested for the ASN being connected. When the value of STATE is 0, this field is ignored.
LOG2_MSG_LENGTH		Message length (in Doublewords) for connected ASN as $\log_2$(MSG_LENGTH). When the value of STATE is 0, this field is ignored.

B2.4.3.4 TX_AMS_CONDIS_ACK

B2.4.3.4.1 Description

This packet is used to acknowledge a TX AMS connect request.

B2.4.3.4.2 Source

AHA

B2.4.3.4.3 Usage constraints

The AAI master must have previously issued a TX AMS connect request that has not yet generated either a response or an error packet.

B2.4.3.4.4 Flow control result

The AAI slave returns a session management token to the AAI master.

B2.4.3.4.5 Fields description

The TX_AMS_CONDIS_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
STATE		This bit identifies the new state. The possible values of this bit are: 0:AMS disconnected. 1:AMS connected When the value of STATE in the unacknowledged TX_AMS_CONDIS_ACK packet is 0, the value of this bit must be 0. When the value of STATE in the unacknowledged TX_AMS_CONDIS_ACK packet is 1, this field can be 0 or 1. 0 normally indicates a serious system configuration failure.
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID initiating this request.
AMS_ID		This field indicates the TX AMS_ID initiating this request.
ERROR		When the value of STATE in the unacknowledged TX_AMS_CONDIS_ACK packet is 1, and STATE value is 0, this field gives the error code. Otherwise ignored.

B2.4.4 Messaging packet group

This section describes the packets belonging to the Messaging packet group.

B2.4.4.1 MSG_SEND

B2.4.4.1.1 Description

This packet is used to convey Revere-AMU packets. Each such data packet can transport one and only one Revere-AMU packet.

B2.4.4.1.2 Source

AMU/AHA

B2.4.4.1.3 Usage constraints

The AAI master or slave can only send a data packet when:

- The channel is in the CONNECTED state.
- The AMI is in the ENABLED state.
- The TX AMS is in the CONNECTED state.
- The TX AMS has enough credits to send the full packet.

B2.4.4.1.4 Flow control result

The TX AMS sends required number of credits to RX AMS.

The requirement to be able to send is :

$AvailableCredits \leq \frac{MessageSize}{HW\_CRED\_SIZE} \leq MAX\_CRED\_USE$

The number of credits consumed is

$\max(MIN\_CRED\_USE, \frac{MessageSize}{HW\_CRED\_SIZE})$

B2.4.4.1.5 Fields description

The MSG_SEND bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the TX AMS_ID targeted by this request. When DIR=0, this field indicates the originating RX AMS_ID.
PRIO		This field is used to set the priority for data packets sent through this AMS
MSG		This field contains the Revere-AMU packet as seen by software

B2.4.4.2 MSG_SEND_ACK

B2.4.4.2.1 Description

This packet is used to return credits over Revere sessions. One such packet can send back credits from several received data packets, so the number of received data packets does not need to match the number of acknowledges sent. However, there cannot be more acknowledges than data packets received.

B2.4.4.2.2 Source

AMU/AHA

B2.4.4.2.3 Usage constraints

The TX AMS must still be in CONNECTED state. The number of returned credits must correspond to additional space available in RX AMS.

B2.4.4.2.4 Flow control result

The RX AMS returns a number of credits to the TX AMS.

B2.4.4.2.5 Fields description

The MSG_SEND_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields

Field name	bits	description
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the TX AMS_ID targeted by this request. When DIR=0, this field indicates the originating RX AMS_ID.
CRED		The value of this field is the number of credits returned

B2.4.5 Credits management packet group

This section describes the Credits management packet group.

B2.4.5.1 CRED_REQ

B2.4.5.1.1 Description

This packet is used to request credits in a Revere ASN from the RX AMS.

B2.4.5.1.2 Source

AMU/AHA

B2.4.5.1.3 Usage constraints

The AAI master or slave can only send a credits request when:

- The channel is in the CONNECTED state.
- The AMI is in the ENABLED state.
- The TX AMS is in the CONNECTED state.

B2.4.5.1.4 Flow control result

None

B2.4.5.1.5 Fields description

The CRED_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the RX AMS_ID targeted by this request.

Field name	bits	description
		When DIR=0, this field indicates the originating TX AMS_ID.
CRED		The value of this field is the number of credits requested. It is a hint.

B2.4.5.2 CRED_REQ_ACK

B2.4.5.2.1 Description

This packet is used to grant credits on Revere sessions following a request.

B2.4.5.2.2 Source

AMU/AHA

B2.4.5.2.3 Usage constraints

The RX AMS must still be in CONNECTED state. The number of returned credits must correspond to space available in RX AMS.

B2.4.5.2.4 Flow control result

The RX AMS grants messaging credits to the TX AMS.

B2.4.5.2.5 Fields description

The CRED_REQ_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the TX AMS_ID targeted by this request. When DIR=0, this field indicates the originating RX AMS_ID.
CRED		The value of this field is the number of credits granted.

B2.4.5.3 CRED_RETURN

B2.4.5.3.1 Description

This packet is used to return credits in a Revere ASN to the RX AMS.

B2.4.5.3.2 Source

AMU/AHA

B2.4.5.3.3 Usage constraints

The AAI master or slave can only return credits when:

- The channel is in the CONNECTED state.
- The AMI is in the ENABLED state.
- The TX AMS is in the CONNECTED state.

B2.4.5.3.4 Flow control result

The TX AMS loses all the messaging credits being returned. None shall remain in the TX AMS.

B2.4.5.3.5 Fields description

The CRED_RETURN bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the RX AMS_ID targeted by this request. When DIR=0, this field indicates the originating TX AMS_ID.
CRED		The value of this field is the number of credits returned.

B2.4.5.4 CRED_RETURN_ACK

B2.4.5.4.1 Description

This packet is used to acknowledge credits return on Revere sessions.

B2.4.5.4.2 Source

AMU/AHA

B2.4.5.4.3 Usage constraints

The RX AMS must still be in CONNECTED state.

B2.4.5.4.4 Flow control result

None.

B2.4.5.4.5 Fields description

The CRED_RETURN_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the TX AMS_ID targeted by this request. When DIR=0, this field indicates the originating RX AMS_ID.

B2.4.5.5 CRED_RECALL

B2.4.5.5.1 Description

This packet is used to recall all credits in a Revere ASN.

B2.4.5.5.2 Source

AMU/AHA

B2.4.5.5.3 Usage constraints

The AAI master or slave can only send a credit recall request when:

- The channel is in the CONNECTED state.
- The AMI is in the ENABLED state.
- The RX AMS is in the CONNECTED state.

B2.4.5.5.4 Flow control result

None

B2.4.5.5.5 Fields description

The CRED_RECALL bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.

Field name	bits	description
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the TX AMS_ID targeted by this request. When DIR=0, this field indicates the originating RX AMS_ID.

B2.4.5.6 CRED_RECALL_ACK

B2.4.5.6.1 Description

This packet is used to recall credits on Revere sessions.

B2.4.5.6.2 Source

AMU/AHA

B2.4.5.6.3 Usage constraints

The TX AMS must still be in CONNECTED state. The number of returned credits must correspond to additional space available in RX AMS.

B2.4.5.6.4 Flow control result

The RX AMS returns all remaining credits to the TX AMS.

B2.4.5.6.5 Fields description

The CRED_RECALL_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the TX AMS_ID targeted by this request. When DIR=0, this field indicates the originating RX AMS_ID.
CRED		The value of this field is the number of credits returned.

B2.4.5.7 CRED_SEND

B2.4.5.7.1 Description

This packet is used to grant messaging credits over a Revere ASN.

B2.4.5.7.2 Source

AMU/AHA

B2.4.5.7.3 Usage constraints

The AAI master or slave can only send a credit recall request when:

- The channel is in the CONNECTED state.
- The AMI is in the ENABLED state.
- The RX AMS is in the CONNECTED state.

B2.4.5.7.4 Flow control result

The connected TX AMS can gain messaging credits.

B2.4.5.7.5 Fields description

The CRED_SEND bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the TX AMS_ID targeted by this request. When DIR=0, this field indicates the originating RX AMS_ID.
CRED		The value of this field is the number of credits granted.

B2.4.5.8 CRED_SEND_ACK

B2.4.5.8.1 Description

This packet is used to recall credits on Revere sessions.

B2.4.5.8.2 Source

AMU/AHA

B2.4.5.8.3 Usage constraints

The TX AMS must still be in CONNECTED state. The number of returned credits must correspond to additional space available in RX AMS.

B2.4.5.8.4 Flow control result

The TX AMS returns granted credits that are not accepted.

B2.4.5.8.5 Fields description

The CRED_SEND_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
DIR		This field is used to resolve LOCAL_AMI_ID and AMS_ID fields
LOCAL_AMI_ID		This field depends on DIR value: When DIR=1, this field indicates the LOCAL_AMI_ID targeted by this request. When DIR=0, this field indicates the originating LOCAL_AMI_ID.
AMS_ID		This field depends on DIR value: When DIR=1, this field indicates the TX AMS_ID targeted by this request. When DIR=0, this field indicates the originating RX AMS_ID.
CRED		The number of messaging credits rejected by the TX AMS.

B2.4.6 DMA management packet group

This section describes the DMA management packet group.

B2.4.6.1 DMA_BME_REQ

B2.4.6.1.1 Description

This packet is used to propagate the PCI Express BME value to AHAs.

B2.4.6.1.2 Source

AMU

B2.4.6.1.3 Usage constraints

The AAI master can only propagate BME when the channel is in the CONNECTED state and the target AMI is in ENABLED state.

B2.4.6.1.4 Flow control result

The AAI master sends a DMA management token to the AAI slave.

B2.4.6.1.5 Fields description

The DMA_BME_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.

Field name	bits	description
BME		This field indicates the new BME value.

B2.4.6.2 DMA_BME_ACK

B2.4.6.2.1 Description

This packet is used to acknowledge a BME propagation request.

B2.4.6.2.2 Source

AHA

B2.4.6.2.3 Usage constraints

The AAI master must have previously issued a BME propagation request that has not yet generated a response.

B2.4.6.2.4 Flow control result

The AAI slave returns a DMA management token to the AAI master.

B2.4.6.2.5 Fields description

The DMA_BME_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.

B2.4.6.3 DMA_TRANS_PEND_REQ

B2.4.6.3.1 Description

This packet is used to request the Transaction Pending state from AHAs.

B2.4.6.3.2 Source

AMU

B2.4.6.3.3 Usage constraints

The AAI master can only issue this request when the channel is in the CONNECTED state and the target AMI is in ENABLED state.

B2.4.6.3.4 Flow control result

The AAI master sends a DMA management token to the AAI slave.

B2.4.6.3.5 Fields description

The DMA_TRANS_PEND_REQ bit assignments are:

Field name	bits	description
MST_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.

B2.4.6.4 DMA_TRANS_PEND_ACK

B2.4.6.4.1 Description

This packet is used to acknowledge a Transaction Pending request.

B2.4.6.4.2 Source

AHA

B2.4.6.4.3 Usage constraints

The AAI master must have previously issued a Transaction Pending request that has not yet generated a response.

B2.4.6.4.4 Flow control result

The AAI slave returns a DMA management token to the AAI master.

B2.4.6.4.5 Fields description

The DMA_TRANS_PEND_ACK bit assignments are:

Field name	bits	description
SLV_PKT_TYPE		This field identifies the packet type. The value of this field is taken from the list of encodings
LOCAL_AMI_ID		This field indicates the LOCAL_AMI_ID targeted by this request.
TP		This field indicates if there are pending read transactions on DMA interface belonging to LOCAL_AMI_ID

B2.4.7 Error Codes

Error codes will be detailed in an upcoming release.

B2.5 AAI Transport Layer

B2.5.1 Introduction

The AAI is based on the unidirectional AXI4-Stream Interface. Therefore, to support bidirectional communication, the AAI transport consists of an AXI4-Stream Interface in each direction, that is:

- A downstream AXI4-Stream Interface connecting an AMU to an AHA. On this interface, the AMU interface is the master and the AHA is a slave.
- An upstream AXI4-Stream Interface connecting an AHA to an AMU. On this interface, the AHA interface is the master and the AMU is a slave.

The AMBA® 4 AXI4-Stream Specification defines a packet as a group of bytes that are transported together across an AXI4-Stream interface. An interconnect between an AHA and an AMU interface must ensure that the stream packet sequence is transferred over the stream protocol interface in the same order in which it was created

B2.5.2 Signals

The interface requires a global clock, ACLK, and a reset signal, ARESETn.

For the AAI, each stream interface is identified by a prefix to the AXI-4 signal names:

- Downstream signals from an AMU to the AHA interface are prefixed with the letters AMU.
- Upstream signals from an AHA interface to an AMU are prefixed with the letters AHA.

Table B2.31: AMU to downstream AHA interface

Signal	Description
AMUTVALID	When set to 1, this signal indicates that the master is driving a valid transfer
AMUTREADY	When set to 1, this signal indicates that the slave can accept a transfer in the current cycle.
AMUTDATA[BN:0]	The interface data path.
AMUTLAST	When set to 1, this signal indicates the final transfer of a packet.
AMUTDEST[N:0]	This signal identifies the target interface to provide routing information for the stream This signal is optional.
AMUTID[N:0]	This signal identifies the originating interface, to provide routing information for the stream. This signal is optional.

Table B2.32: AHA to upstream AMU interface

Signal	Description
AHATVALID	When set to 1, this signal indicates that the master is driving a valid transfer
AHATREADY	When set to 1, this signal indicates that the slave can accept a transfer in the current cycle.
AHATDATA[BM:0]	The interface data path.
AHATLAST	When set to 1, this signal indicates the final transfer of a packet.

Signal	Description
AHATDEST[N:0]	This signal identifies the target interface to provide routing information for the stream This signal is optional.
AHATID[N:0]	This signal identifies the originating interface, to provide routing information for the stream. This signal is optional.

In the above tables:

- BN and BM are the numbers associated with the most significant bit on a datapath that is required to be an integral number of bytes wide.
- N is the value log(base2) M rounded up to the nearest integer, where M is the maximum number of endpoints supported by the AMU implementation.
- Values of TDEST and TID must be allocated sequentially without gaps, in order of ascending affinity

For further information about the signals used by the AMU AHA Interface, and for details about handshaking, see AMBA® 4 AXI4-Stream Protocol Specification [8].

Chapter B3
AHA DMA Transport Layer

The AHA may have an optional DMA interface to access out-of-band data. If present, such interface must comply with the following rules:

- ACE5-lite compliant
- Support for Untranslated_Transactions extension
- Optional support for Cache_Stash_Transactions extension

as per AMBA® AXI and ACE Protocol Specification [4].

The number of interfaces per hardware agent (AHA) is IMPLEMENTATION DEFINED.

Chapter B4
Power control

To be added in future release.

Part C
Use cases

Chapter C1
Null accelerator

C1.1 Introduction

This example shows how to send/receive messages to/from a simple AHA using Message Format 0 (MFO0) messages with in-band data.

The device comprises a single AMU with a *null accelerator AHA*. This AHA has the following characteristics:

- It supports two hardware contexts
- Each hardware context has an AMI-HW with a single RX AMS (request) and a single TX AMS (response)
- It has no DMA resource and each context has no associated state

The Function of the null accelerator is to simply return any request message on the RX AMS as a response on the TX AMS.

Note

The null accelerator template can be modified to support a range of simple lookaside accelerator functions. The request message contains any necessary input data and the response message contains a result.

The null accelerator example shows how the two accelerator contexts are used by two separate user drivers. Each software driver has an AMI-SW with two ASNs (request and response) connected to an AMI-HW on the null accelerator.

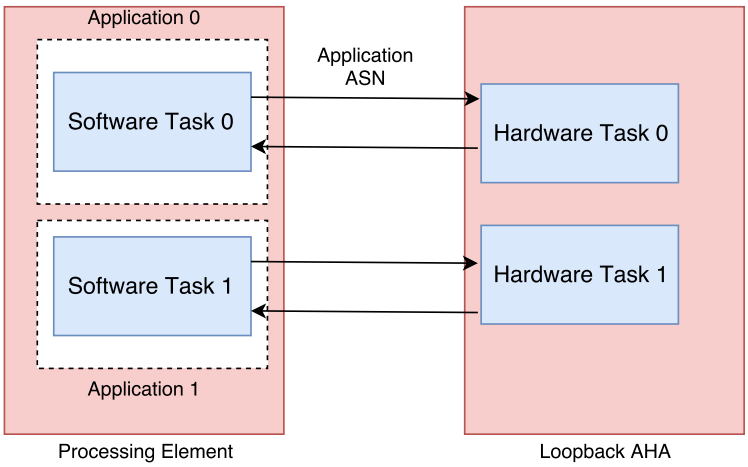

Figure C1.1: Logical view of the system showing the configured ASNs between the HW accelerator contexts and software drivers.

C1.2 Software architecture

In this example there are two software drivers:

One instance of a PF driver that configures the AMU including statically setting up the required ASNs. There is no AHA specific configuration for the null accelerator. Any such configuration would be included in the PF driver.

Two instances of a combined VF and user driver that do both VF specific configuration and use the device by sending requests and receiving responses. These drivers could be contained in separate virtual machines.

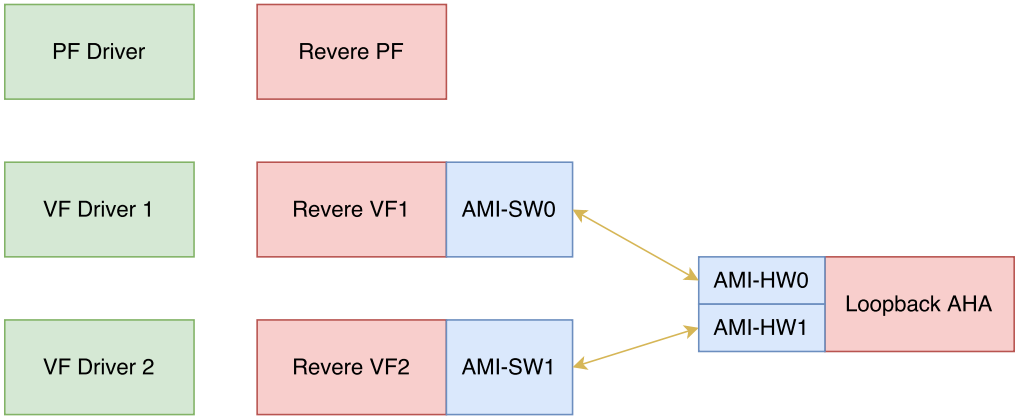

Figure C1.2: View of the complete system showing the mapping of software drivers and AMI-SWs to PCI Functions and ASNs configured between AMI-SWs and the null accelerator AHA

Note

In the picture, AMI IDs are physical, meaning they belong to a global ID space of AMIs that are implemented by the AMU. As it is explained as part of the PF driver and VF driver operations in future sections, AMIs are indexed locally within a specific Function for configuration purposes.

The device PF driver uses the management interface to manage the overall configuration of the system (including ASN creation and mapping of AMI-SW to Functions).

The device VF driver 1 and VF driver 2 use the management interface to manage the configuration for VF1 and VF2 respectively. Device VF driver 1 and VF driver 2 also make use of physical AMI-SW0 and AMI-SW1 respectively to communicate with the first and second contexts of the null accelerator AHA (Request/Response).

Note

As detailed below in Table C1.1 and Table C1.2, all AMI-SWs and AMI-HWs are mapped with local ID zero in the two VFs.

C1.3 PF driver operation

C1.3.1 Binding the PF driver to the PF

A PF driver running on the host machine is bound to the Physical Function of the device (AMU and associated AHA).

System software binds the PF driver by matching the Device ID of the Physical Function. Once the PF driver is loaded, it reads the capabilities registers and performs initialisation.

C1.3.2 Virtual Function and AMI allocation

The second step during the initialisation process is to apportion the system resources to Virtual Functions.

This example utilises two Virtual Functions with one AMI-SW each. The PF driver performs the following steps to create and map the AMI-SW to the Virtual Functions:

1. Use the PCI architected mechanisms to configure two Virtual Functions. The operating system typically provides an abstraction API for these mechanisms.
2. Configure the management interface. This includes configuring appropriate TX and RX interrupt vectors as well as allocating memory for the ring slots and configuring the base pointers appropriately.
3. Enable the AMU by setting the AMU_CR.AMU_EN bit in the management interface.
4. Request the creation of two AMI-SWs and two AMI-HWs through the management AMI.

To achieve the last step, the PF driver needs to send appropriate commands through the command AMS and wait for responses coming through the response AMS in the management AMI. The following sequence diagram shows the messages that the PF driver exchanges with the AMU in order to perform this configuration:

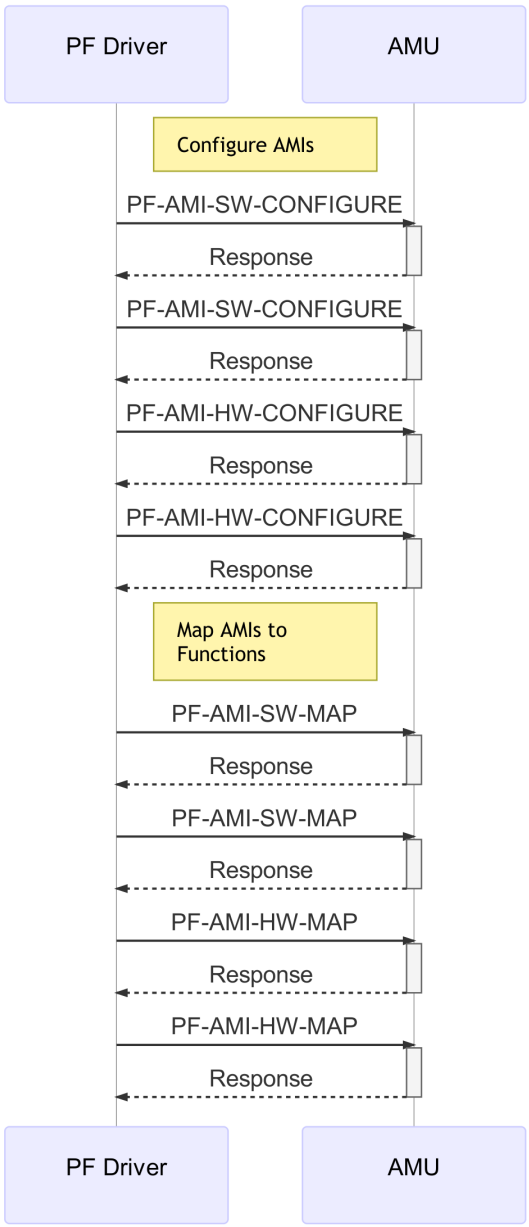

Figure C1.3: PF-VF Driver Communication Sequence diagram.

The following table shows the parameters used to map and configure the two AMI-SWs:

Table C1.1: AMI-SW mapping

Physical AMI-SW	Function Owner	AMI-SW ID	PASID
0	1	0	-
1	2	0	-

The following table shows the parameters used to map and configure the two AMI-HWs:

Table C1.2: AMI-HW mapping

Physical AMI-HW	AHA ID	Function Owner	AMI-HW ID	PASID	LOG2_HW_CRED_SIZE
0	0	1	0	-	1
1	0	2	0	-	1

Note

The AMIs are not yet enabled at this point

C1.3.3 ASN allocation

Note that, at this point, no ASN has been established. The following ASN are required per AMI-SW for this example:

- An ASN from the AMI-SW to the null AHA. This is used by software to send messages to the null AHA.
- An ASN from the null AHA to the AMI-SW. This is used by software to receive messages from the null AHA.

Note

In this example, the role and number of ASN per AMI is known in advance, therefore it is possible for the PF driver to configure all needed ASN at initialisation time. For other examples, the VF driver sends IMPLEMENTATION DEFINED messages to the PF driver via the AMU to request a specific configuration.

The PF driver requests the creation of ASNs through the management interface. The following sequence diagram shows the messages that the PF driver exchanges with the AMU in order to perform this configuration:

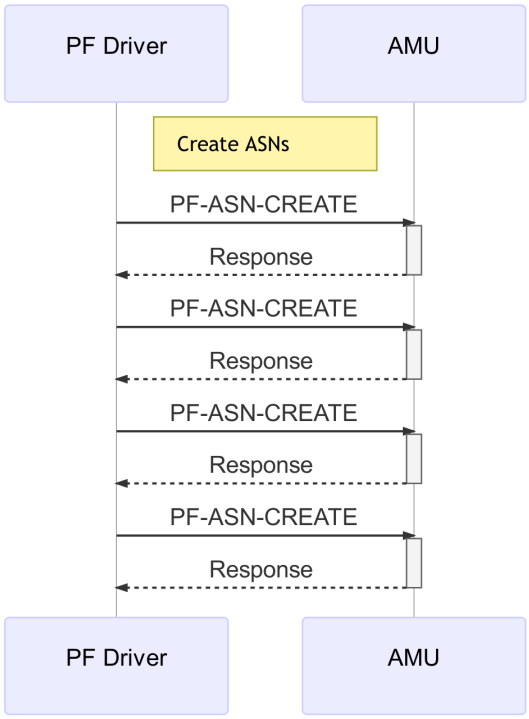

Figure C1.4: PF-VF Driver Communication Sequence diagram.

The following table shows the parameters used to create the four ASNs:

ASN ID	Message Format	Src. Owner	Src. AMI IDX	Src. AMS	Dst. Owner	Dst. AMI IDX	Dst. AMS
0	0	1	AMI-SW0	0	1	AMI-HW0	0
1	0	1	AMI-HW0	0	1	AMI-SW0	0
2	0	2	AMI-SW0	0	2	AMI-HW0	0
3	0	2	AMI-HW0	0	2	AMI-SW0	0

Note

Stashing, profiling and tracing are not required and not used in this example.

The following diagram shows the AMS and ASN that have been created.

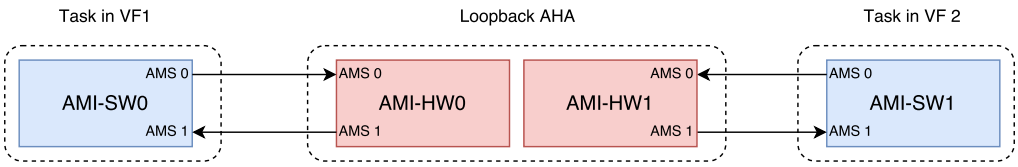

Figure C1.5: Sessions created for the Loopback example.

C1.4 VF driver operation

C1.4.1 Binding the VF and user driver to the VF

A VF driver running on the host machine is bound to the Virtual Function of the device (AMU and associated AHA).

System software binds the combined VF and user driver by matching the Device ID of the Virtual Function. Once the VF driver is loaded, it reads the capabilities registers and performs initialisation.

System software provides mechanisms for assignment of a VF to a virtual machine or to a driver running in user-space. An example is the VFIO framework in Linux.

C1.4.2 AMS Allocation

The PF driver has appropriately configured AMIs and ASNs required for this example. The last step to get the ASN to a functional state is to configure the AMS appropriately. This is the responsibility of the VF driver.

Each driver must allocate sufficient AMSs; in this case, a total of one TX AMS and one RX AMS per AMI-SW. The following diagram shows the messages that each VF driver exchanges with the AMU to configure the AMSs.

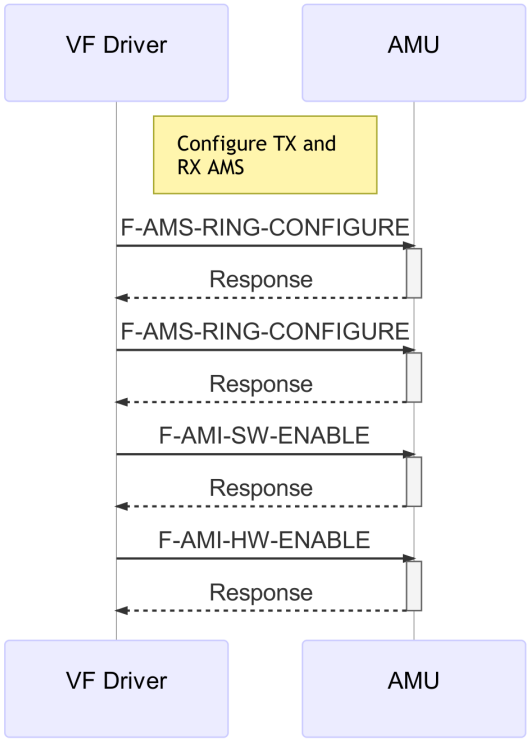

Figure C1.6: PF-VF Driver Communication Sequence diagram.

The following table shows example parameters used to configure the AMSs:

Table C1.4: AMI-SW configuration parameters

LOG2_SLOT_SIZE	LOG2_SIZE	RX_MODE	THRESHOLD	RING_BASE_PTR
3	3	BACK_PRESSURE	0	-

The RING_BASE_PTR must be set according to the translation scheme that system software configures for the Virtual Function.

The index must be initialised to 0. The remaining parameters are not critical for the purposes of this example and can be modified as needed. The following have been chosen:

- A $\log_2$ slot size of 3 provides 64 bytes per slot (corresponding to a maximum of 60 bytes of payload) and sets an upper bound for the message size that can be sent/received through this AMS.
- A $\log_2$ size of 3 provides 8 slots in this AMS.
- The back pressure mode provides guaranteed delivery.
- A threshold of 0 means that the relevant digest will be modified as soon as a slot is freed up in a previously full TX AMS or a message arrives in a previously empty RX AMS.

Each of the Virtual Function drivers can now access the AMI-SW interfaces assigned to them through the VF BAR space.

C1.4.3 AMS usage

At this stage the system has been fully configured and either of the VF drivers can send and receive messages since AMIs are enabled.

The following snippets of code show examples for sending and receiving through an A1 socket type.

In Listing C1.1, the ring is first checked for empty spaces. The slot is pointed to the appropriate location in the ring slot array. Once the slot has been filled, a memory barrier is issued and the write index is updated.

Listing C1.1: Enqueuing a message to an AMS-TX

```
if ((unsigned int)(WRITE_INDEX - READ_INDEX) == (1U << LOG2_SIZE)) {
    return -1
}

*slot = &base_ptr[(WRITE_INDEX & MASK) << (LOG2_SLOT_SIZE + 3)]

// Fill slot

memory_barrier()
WRITE_INDEX++
```

Where MASK == (1 << LOG2_SIZE) - 1.

In Listing C1.2, the ring is first checked for messages. A pointer to the message in the ring is created. Once the message has been read, a memory barrier is issued and the read index is updated.

Listing C1.2: Dequeueing a message from an AMS-RX

```
if (WRITE_INDEX == READ_INDEX) {
    return -1
}

*slot = &data[(READ_INDEX & MASK) << (LOG2_SLOT_SIZE + 3)]

// Copy contents of message to internal memory
```

```
    memory_barrier()
    READ_INDEX++
```

Where MASK == (1 << LOG2_SIZE) - 1.

When sending a message, the AMU pulls the message from the ring, and transports it to the null accelerator AHA. Similarly, messages generated by the AHA are transported to the appropriate RX AMS. Privileged software is responsible for setting up the ASN (in this case, the PF driver), and lower privileged software can participate in the data path through the pre-established ASN.

When the ring is full, software can choose to give up or retry the operation. When the AMU pulls a message from a TX ring, the crediting system guarantees that the receiver will have enough buffer space.

C1.5 Application

Any message pushed to TX AMS 0 on either of the drivers will reach the null accelerator AHA through their associated AMI-HW, which will loop the message back to RX AMS 0, assigned to the same application.

Chapter C2
VM Live Migration

C2.1 Introduction

This example shows how a Revere-AMU Virtual Function can be migrated from one physical machine to another physical machine.

The device comprises a single AMU with a Null accelerator AHA. This AHA has the following characteristics:

- It supports multiple hardware contexts.
- Each hardware context has an AMI-HW with a single RX AMS (request) and a single TX AMS (response).
- Each context has IMPLEMENTATION DEFINED state.

The behaviour of the AHA and the context associated with an AMI-HW is not important for the purposes of this example.

This example presents a single VF, containing:

- One AMI-SW
- One AMI-HW associated with the Null accelerator AHA.

The AMI-SW has two ASNs (request and response) configured with the AMI-HW on the Null accelerator.

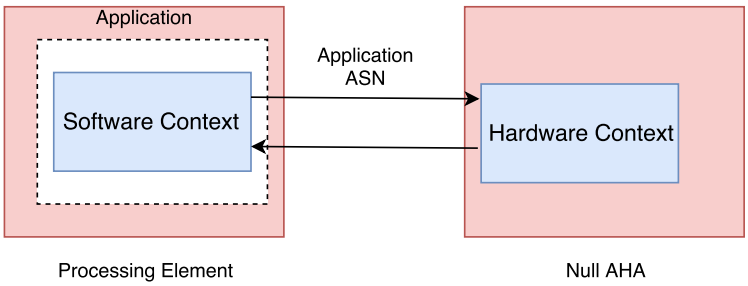

Figure C2.1: Logical view of the system showing the configured ASNs between the HW accelerator contexts and software drivers.

C2.2 Initial setup

In this example there are two software drivers:

One instance of a PF driver that configures the AMU including statically setting up the required ASNs. The AHA also requires IMPLEMENTATION DEFINED per-context configuration. This configuration is also performed by the PF driver.

One instance of a combined VF and user driver that do both VF specific configuration and use the device by sending requests and receiving responses. This driver is contained within a Virtual Machine.

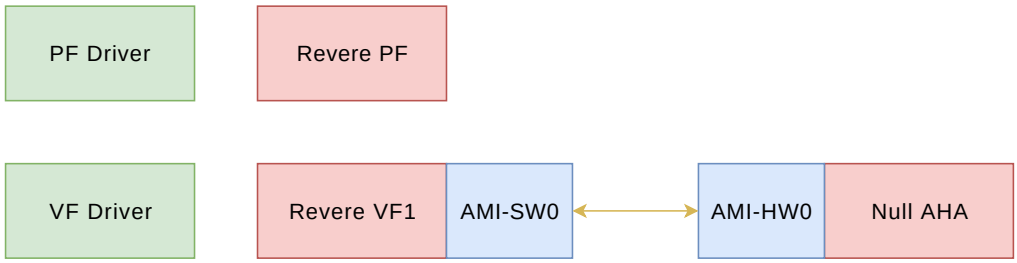

Figure C2.2: View of the complete system showing the mapping of software drivers and AMI-SWs to PCI Functions and ASNs configured between AMI-SWs and the Null accelerator AHA

Note

In the picture, AMI IDs are physical, meaning they belong to a global ID space of AMIs that are implemented by the AMU. When configuring an AMI, the owning Function refers to the AMI using a local ID.

The device PF driver uses the management interface to manage the overall configuration of the system (including ASN creation and assignment of AMI-SW to Functions).

The device VF driver uses the management interface to manage the configuration for VF1. Device VF driver also makes use of AMI-SW0 to communicate with a context in the Null accelerator AHA (Request/Response).

C2.3 Live Migration and device state

Live migration implies relocating a Virtual Machine from one host (source host) to another host (destination host), without the guest Operating System and associated applications noticing that this process is taking place from a functional perspective.

This process is usually triggered by orchestration software running in another physical location and relies on hypervisor software within the source and destination hosts to carry out the necessary operations.

When a Virtual Machine is migrated, all the state associated with it must be transparently transported to the destination host. This state includes:

- All the memory owned by the guest.
- All the state associated with the CPU, such as registers.
- All the state associated with devices used by the guest.

Once all the state has been collected, it is transported and restored in the destination host. The guest OS or the VF driver are not involved at any point in time.

A Revere-AMU device encapsulates the following state:

- All registers exposed by the Virtual Function (see A3.3 *Revere-AMU and PCI Express*).
- AMI mappings. The Virtual Function in the destination VM must have the same number of AMIs mapped in exactly the same locations (see A3.4.4.3 *Mapping AMIs to Functions*).
- AMI-SW state associated with the Virtual Function (see A3.4.4.1.2 *AMI-SW Configuration*).
- AMI-HW state associated with the Virtual Function (see A3.4.4.1.3 *AMI-HW Configuration*).
- ASN state associated with the Virtual Function (see A3.4.4.1.4 *ASN Configuration*).
- AMS state associated with the Virtual Function (see A3.4.4.1.5 *AMS Configuration*).

The Revere-AMU architecture provides the means to retrieve and restore all the architected state associated with a Virtual Function.

A Revere-AMU device may also have non-architected state, including:

- Any state relating to the Virtual Function that is maintained in the PF driver.
- Any state within an AHA.

It is the responsibility of the PF driver to save and restore this state and for AHAs to provide IMPLEMENTATION DEFINED messages/registers for saving and restoring state.

C2.4 Starting point and prerequisites

This example assumes that:

- A VF has been created.
- AMIs have been mapped.
- ASNs between the AMIs have been created.
- Both the PF and VF are bound to a PF driver running on the host and a VF driver running inside a VM respectively.

For the details on how this initial setup is performed, refer to Chapter C1 *Null accelerator*.

The VF driver is making use of the AMI-SW and there are potentially in-flight messages when the live migration process is triggered.

As part of the migration process, the hypervisor saves the state associated with all of the devices that are assigned to the VM. In the case of the Revere-AMU device, the Revere-AMU PF driver is in control of all the state. Because of this, it is required that the hypervisor interacts with the Revere-AMU PF driver for the purposes of triggering device saving and restoration. Whilst it is assumed that the PF driver is on the live migration loop, the mechanism used for this depends on the hypervisor.

In this example, both the PF driver at the source and the PF driver at the destination host are involved in the migration process; the former is responsible for saving the state associated with the running VF, whereas the latter is responsible for restoring the state in one of the available VFs.

Neither the VF driver nor the applications that utilise AMIs mapped to it are involved in the process.

The following sections describe the steps that the PF drivers must carry out to migrate a VF.

C2.5 Source PF driver operation

The PF driver in the source host is responsible for saving the state of the running VF. This process can be divided in the following steps:

- Disabling the migrating VF.
- Saving management registers.
- Saving AMI-SW state.
- Saving AMS state.
- Saving AMI-HW state.
- Saving ASN state.
- Saving non-architected state.
- Unmapping AMI-SWs and AMI-HWs and resetting VF.

Except for the management registers, all of the state is accessed by sending commands through the management AMI which is part of the management interface (see A3.4.4 *Management Messages*).

Note

In addition to the state mentioned above, the PCIe Configuration space Type 0 header and MSI-X structures associated with the Function must be saved and restored. This can usually be done by the hypervisor without the involvement of the device driver.

The following sections show the commands that are required to perform each of these steps.

C2.5.1 Disabling the migrating VF

Depending on the scenario, the VM owning the VF is suspended.

The following diagram shows command/response messages exchanged through the management AMI when disabling a Function.

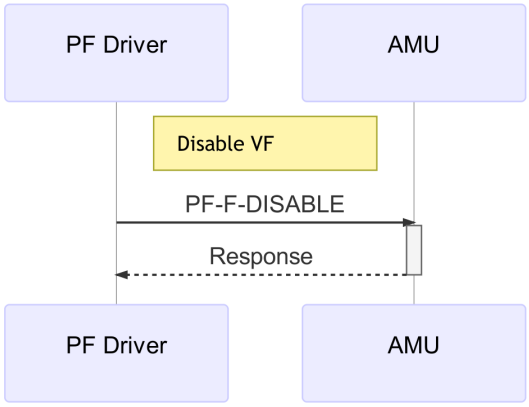

Figure C2.3: PF driver-AMU communication sequence diagram.

Disabling a Virtual Function also quiesces all of the AMIs mapped to it, including the management AMI (see A3.4.4.4 *Enable and disable operations*). This means that, once the response message is received from the AMU, there are no in-flight messages associated with the migrating Function and it is safe to start retrieving all of its state.

After a Function is disabled, there might still be messages buffered in either the TX or RX AMSs. For AMI-SWs, these messages are effectively in memory and will be transported by hypervisor software as part of the memory-related state as a single binary blob. For AMI-HWs, the process would involve retrieving these messages from AHAs in an IMPLEMENTATION DEFINED manner.

C2.5.2 Saving management registers

Revere-AMU Virtual Functions include a set of management registers to control the configuration of the VF as well as to probe certain architected parameters (see A3.4.3 *Management Registers*). Registers that are writable by the VF driver must be saved and restored in the destination host.

When the live migration process is started, hypervisor software must ensure that the guest OS cannot modify the contents of management registers by trapping any accesses performed to the BAR space of the migrating VF.

C2.5.3 Saving AMI-SW state

The following diagram shows command/response messages exchanged through the management AMI when saving AMI-SW state.

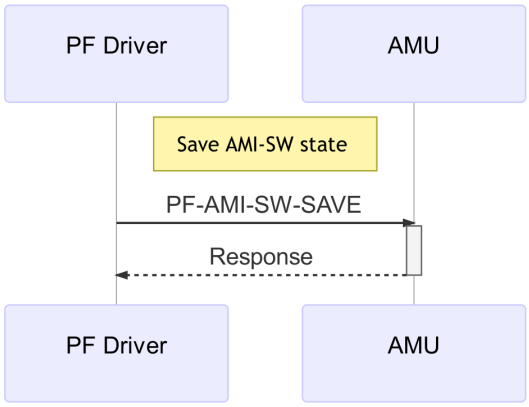

Figure C2.4: PF driver-AMU communication sequence diagram.

Since there is a single AMI-SW mapped to the VF, this process is only done once, for that AMI-SW.

The response message includes all of the state associated with the AMI-SW (see A3.4.4.1.2 *AMI-SW Configuration*). This state must be stored in a way such that hypervisor software can attach it to the binary blob that is eventually transported to the destination host.

C2.5.4 Saving AMS state

The following diagram shows command/response messages exchanged through the management AMI when saving AMS state.

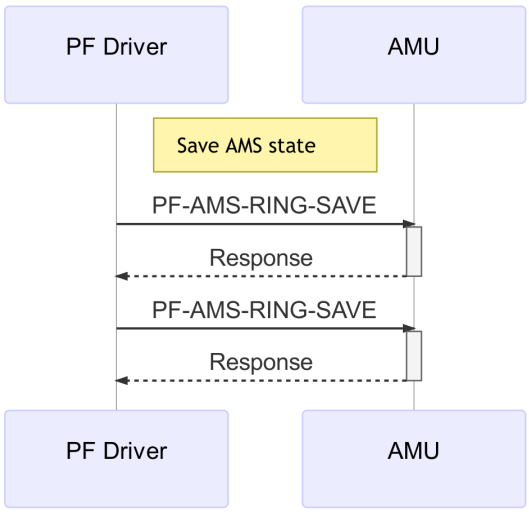

Figure C2.5: PF driver-AMU communication sequence diagram.

In this example, there are two AMSs associated with the AMI. The state associated with each is retrieved separately.

The response message includes all of the state associated with the AMS (see A3.4.4.1.5 *AMS Configuration*). This state must be stored in a way such that hypervisor software can attach it to the binary blob that is eventually transported to the destination host.

C2.5.5 Saving AMI-HW state

The following diagram shows command/response messages exchanged through the management AMI when saving AMI-HW state.

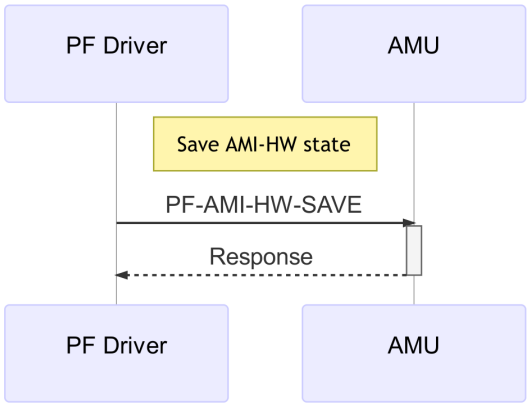

Figure C2.6: PF driver-AMU communication sequence diagram.

Since there is a single AMI-HW mapped to the VF, this process is only done once, for that AMI-HW.

The response message includes all of the state associated with the AMI-HW (see A3.4.4.1.3 *AMI-HW Configuration*). This state must be stored in a way such that hypervisor software can attach it to the binary blob that is eventually transported to the destination host.

C2.5.6 Saving ASN state

The following diagram shows command/response messages exchanged through the management AMI when saving ASN state.

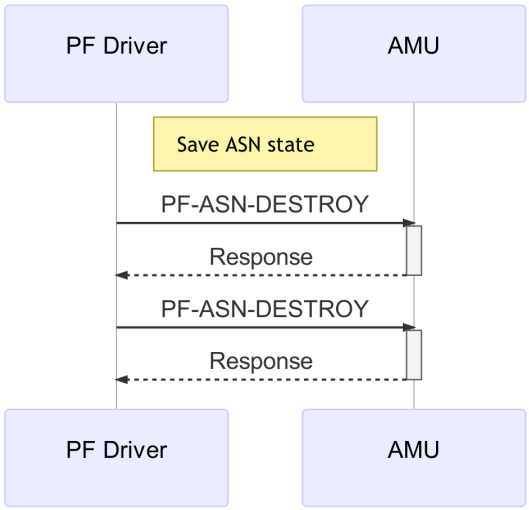

Figure C2.7: PF driver-AMU communication sequence diagram.

Since two ASNs are assumed to be set up in this example, this process is done twice, once for each ASN. Note that, to retrieve the ASN state, the ASN must be destroyed.

The response message includes all of the state associated with the ASN (see A3.4.4.1.4 *ASN Configuration*). This state must be stored in a way such that hypervisor software can attach it to the binary blob that is eventually transported to the destination host.

C2.5.7 Saving indices and PBAs

After Function quiescing phase since the VM is suspended TX and RX queues indices should remain the same. Hypervisor would then save all the TX_DIGEST_MASK,RX_DIGEST_MASK,READ_INDEX,WRITE_INDEX in the AMS data structures from the migrated Function BAR Memory Space aperture.

On the pending MSI-X side, if Bus Master Enable is clear or MSI-X Function Masking is enabled, some bits can be set in the MSI-X PBA data structure of the Function. In that case those bits should be saved and restored in an IMPLEMENTATION DEFINED manner making sure that no spurious MSI-X is sent in the process. Please refer to A3.7.7 *Interrupts* for more details on MSI-X handling in Revere).

C2.5.8 Saving non-architected state

In this example, the AHA is assumed to have specific per-context state as well as, potentially, messages buffered in the AHA. Revere-AMU does not architect a way to retrieve such state. It is IMPLEMENTATION DEFINED how device-specific or AHA-buffered messages are retrieved by the PF driver.

The PF driver may also have IMPLEMENTATION DEFINED state associated with the VF. It is the responsibility of the PF driver to save this state.

All this data is added to a minimal subset of the execution state of the VM to be sent to target host.

C2.5.9 Unmapping AMI-SWs and AMI-HWs and resetting VF

At this stage the PF driver may release all of the resources associated with the migrating VF. The following diagram shows command/response messages exchanged through the management AMI when releasing these resources.

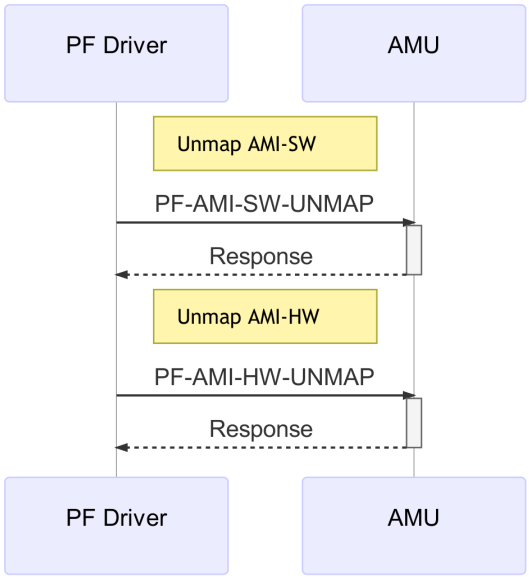

Figure C2.8: PF driver-AMU communication sequence diagram.

After migration is completed, it is advisable to perform an FLR on the VF.

C2.6 Destination PF driver operation

The PF driver in the destination host is responsible for restoring the state in the new VF. This process can be divided in the following steps:

- Resetting VF and mapping AMI-SWs and AMI-HWs.
- Restoring non-architected state.
- Restoring ASN state.
- Restoring AMI-HW state.
- Restoring AMI-SW state.
- Restoring AMS state.
- Restoring management registers.
- Enabling the new VF.

Except for the management registers, all of the state is accessed by sending commands through the management AMI which is part of the management interface (see A3.4.4 *Management Messages*).

Note

In addition to the state mentioned above, the PCIe Configuration space Type 0 header and MSI-X structures associated with the Function must be saved and restored. This can usually be done by the hypervisor without the involvement of the device driver.

The following sections show the commands that are required to perform each of these steps.

C2.6.1 Resetting VF and mapping AMI-SWs and AMI-HWs

The PF driver needs to allocate a suitable VF. Before migration is started, software should perform an FLR on the destination VF.

The PF driver then needs to map the AMI-SWs and AMI-HWs that the VF requires. AMI-SWs must be mapped in the same location used in the source host. The following diagram shows command/response messages exchanged through the management AMI when mapping AMIs.

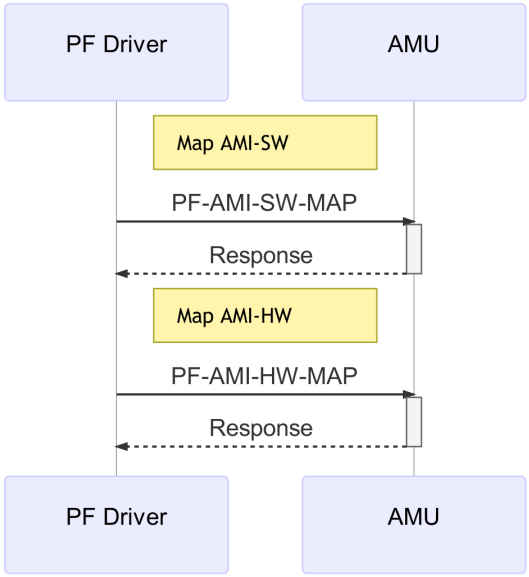

Figure C2.9: PF driver-AMU communication sequence diagram.

C2.6.2 Restoring indices and PBAs

On the migration target AMS indices are restored by writing to the Function BAR Memory Space aperture, which should not generate spurious interrupts since the AMIs owning Function is disabled.

If required MSI-X PBA bits are also restored in an IMPLEMENTATION DEFINED manner making sure that no spurious MSI-X is sent in the process.

C2.6.3 Restoring non-architected state

In this example, the AHA is assumed to have specific per-context state, as well as, potentially, messages buffered in the AHA. Revere-AMU does not architect a way to restore such state. It is IMPLEMENTATION DEFINED how device-specific or AHA-buffered messages are restored by the PF driver.

The PF driver may also have IMPLEMENTATION DEFINED state associated with the VF. It is the responsibility of the PF driver to restore this state.

C2.6.4 Restoring ASN state

The following diagram shows command/response messages exchanged through the management AMI when restoring ASN state.

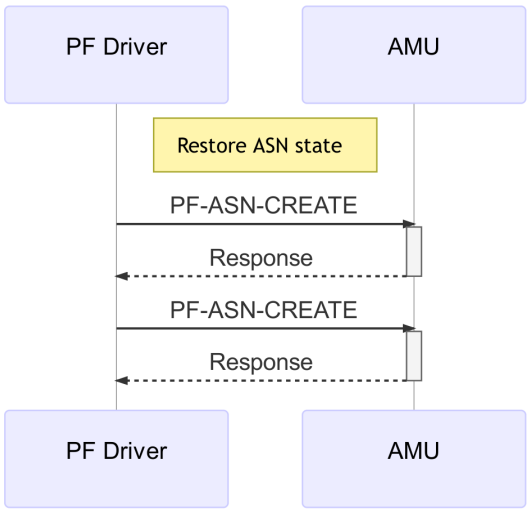

Figure C2.10: PF driver-AMU communication sequence diagram.

Since two ASNs are assumed to be set up in this example, this process is done twice, once for each ASN. Note that ASN state is restored when the ASN is created (see A3.4.4.1.4 *ASN Configuration*).

C2.6.5 Restoring AMI-HW state

The following diagram shows command/response messages exchanged through the management AMI when restoring AMI-HW state.

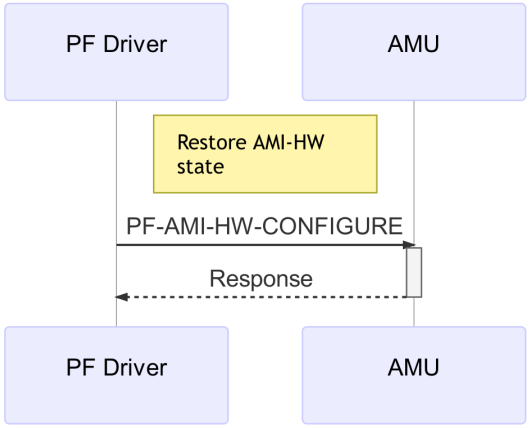

Figure C2.11: PF driver-AMU communication sequence diagram.

Since there is a single AMI-HW mapped to the VF, this process is only done once, for that AMI-HW.

C2.6.6 Restoring AMI-SW state

The following diagram shows command/response messages exchanged through the management AMI when restoring AMI-SW state.

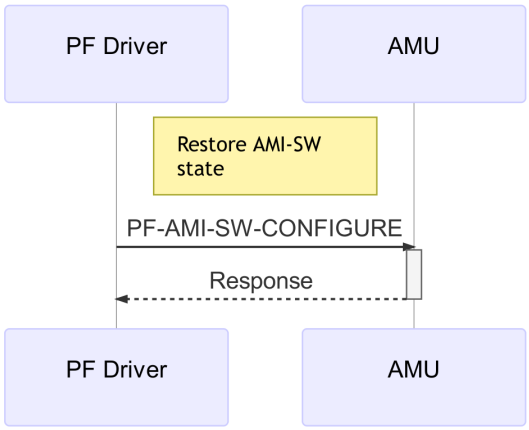

Figure C2.12: PF driver-AMU communication sequence diagram.

Since there is a single AMI-SW mapped to the VF, this process is only done once, for that AMI-SW.

C2.6.7 Restoring AMS state

The following diagram shows command/response messages exchanged through the management AMI when restoring AMS state.

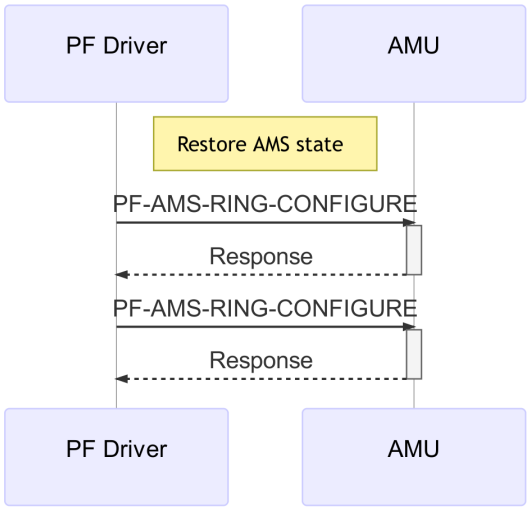

Figure C2.13: PF driver-AMU communication sequence diagram.

In this example, there are two AMSs associated with the AMI. The state associated with each is restored separately.

C2.6.8 Restoring management registers

Revere-AMU Virtual Functions include a set of management registers to control the configuration of the VF as well as to probe certain architected parameters (see A3.4.3 *Management Registers*). Registers that are writable by the VF driver must be saved and restored in the destination host.

C2.6.9 Enabling the new VF

The following diagram shows command/response messages exchanged through the management AMI when enabling a Function.

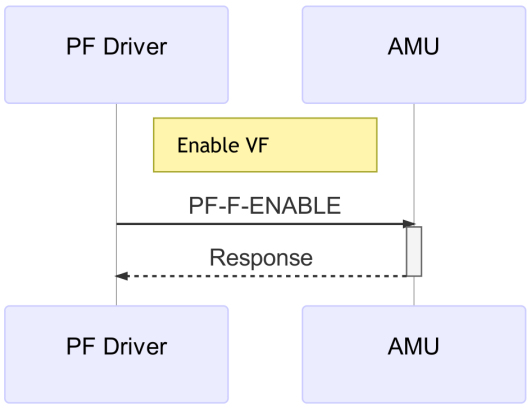

Figure C2.14: PF driver-AMU communication sequence diagram.

Chapter C3
Overprovisioning

C3.1 Introduction

This example shows how a Revere-AMU AMI-SW can be overprovisioned by more privileged software to present more AMIs to software than what is effectively supported by the hardware.

The device comprises a single AMU with a Null accelerator AHA. This AHA has the following characteristics:

- It supports one hardware context.
- The hardware context has an AMI-HW with a single RX AMS (request) and a single TX AMS (response).
- The context has an IMPLEMENTATION DEFINED state.

The behaviour of the AHA and the context associated with an AMI-HW is not important for the purposes of this example.

This example presents two VFs (VF1 and VF2), mapped onto two VMs (VM1 and VM2), and a hypervisor running the PF driver.

The AMI-HW associated with the Null accelerator AHA and the AMI-SW are mapped at one point in time to only one of the VFs.

Both AMIs have two ASNs (request and response). We assume that the AMI-SW is only used by the PE running the VM.

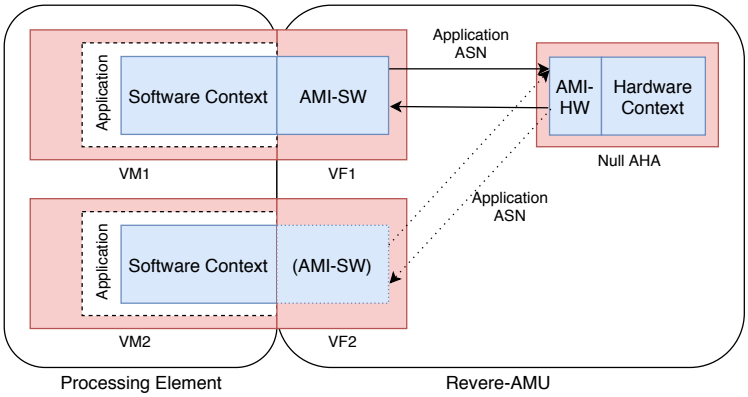

Figure C3.1: Logical view of the system showing the configured ASNs between the HW accelerator context and software drivers for hypervisor level overprovisioning.

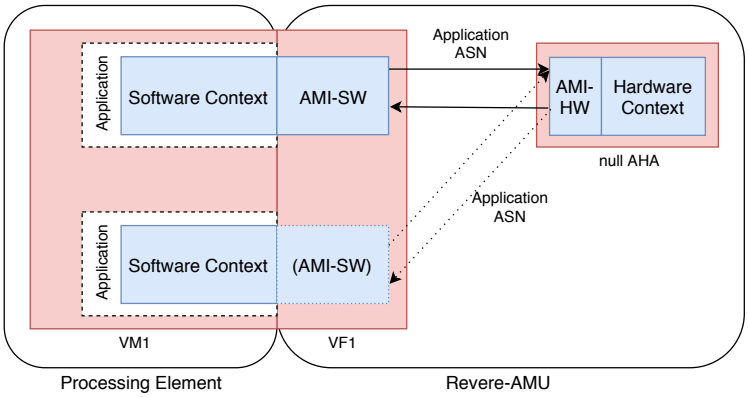

Figure C3.2: Logical view of the system showing the configured ASNs between the HW accelerator context and software drivers for OS level overprovisioning.

C3.2 Initial setup

In this example there are two software drivers:

One instance of a PF driver that configures the AMU including statically setting up the required ASNs. The AHA also requires IMPLEMENTATION DEFINED per-context configuration. This configuration is also performed by the PF driver.

Two instances of a combined VF and user driver that do both VF specific configuration and use the device by sending requests and receiving responses. This driver is contained within each Virtual Machine.

The device PF driver uses the management interface to manage the overall configuration of the system (including ASN creation and assignment of AMIs to Functions).

The VM1 device VF driver uses the management interface to manage the configuration for VF1, the VM2 device VF driver uses the management interface to manage the configuration for VF2. In this scenario device VF drivers make use alternatively of the AMI-SW to communicate with the context in the Null accelerator AHA (Request/Response).

C3.3 Overprovisioning rationale

Some application systems might need more AMI-SW than what their Revere-AMU implementation can provide, each AMI-SW not being fully utilized at all times. Overprovisioning allows a more privilged software to present more AMI-SW to software applications than what are actually available. This use case provides two simple examples of how this could be achieved through transparent AMI remapping between two applications running in different VMs (first example) or in the same (sometimes virtualized) OS.

C3.4 Starting point and prerequisites

This example assumes that:

- One (OS overprovisioning) or two (hypervisor overprovisioning) VFs have been created.
- Both AMIs (AMI-SW and AMI-HW) have been mapped to VF1 as a starting point.
- ASNs between the AMI-SW and AMI-HW have been created.
- The PF and VFs are bound to a PF driver running on the host and VF drivers running inside the VMs.

For the details on how this initial setup is performed, refer to Chapter C1 *Null accelerator*.

We assume in this use case that application usage of the AHA context is sparse and thus AMI-HW can be shared without noticeable performance penalty.

C3.5 Operation sequence

For the hypervisor overprovisioning example, let us assume the PF driver gets a request from the VF driver in VM2 to setup a connection to use the null AHA context.

The hypervisor has a mechanism to schedule AMIs across Virtual Functions based on a use-case specific policy.

An example of such policy could be inactivity. In that case, example mechanisms that could be used to monitor activity could be:

- Un-mapping AMI-SW data structures from stage 2 tables used by the MMU and trap accesses.
- Reset access flag AF for the page containing AMI-SW data structures and monitor it.
- Use Revere-AMU counters to monitor AMI-SW activity.

At some point, the hypervisor decides to remap an AMI-SW from VF1 (assigned to VM1) into VF2 (assigned to VM2) following an application's request in VM2.

In that case the PF driver could try to remap the AMI-SW and AMI-HW from VF1 to VF2 while monitoring VM1 application usage of the AMI-SW.

We assume the PF driver takes appropriate steps to trap transactions to AMI-SW and can potentially abort overprovisioning in such event.

Note

It is IMPLEMENTATION DEFINED whether a potential abort and revert can happen during the transition phase or at the end.

After AMI relocation happened any access to AMI-SW from VM1 application can be considered as an overprovisioning scenario.

Note

The hypervisor also needs to trap configuration messages targeting the remapped AMI. For this purpose, it enables trapping those messages using the PF-F-TRAP-CONFIGURE message.

Similarly for the OS overprovisioning example, let us assume the second application in VM1 needs to use the AMI-SW. The VM1 OS could monitor such need by:

- Un-mapping AMI-SW data structures from stage 1 tables used by the MMU and trap accesses.
- Reset access flag AF for the page containing AMI-SW data structures and monitor it.
- Use Revere-AMU counters to monitor AMI-SW activity.

At this point, the OS (VM1 guest OS here) decides to remap the AMI-SW from application 1 process space into application 2 process space.

Comments from the previous example also apply here. This is just simpler since it does not involve AMI remapping nor hypervisor intervention.

C3.5.1 Disabling and saving the remapped AMI-HW

The following diagram shows command/response messages exchanged through the management AMI when disabling the remapped AMI-HW.

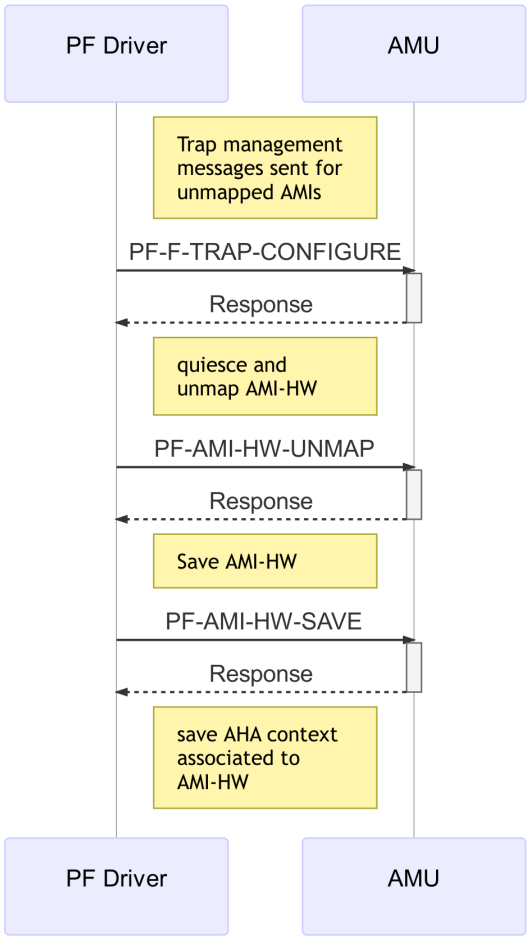

Figure C3.3: PF driver-AMU communication sequence diagram to prepare relocation of AMI-HW.

It is IMPLEMENTATION DEFINED how the PF driver will save the AHA context associated to the AMI-HW being remapped. Quiescing the AMI-HW might keep messages in the AMI-HW context. Such messages should also be saved as part of IMPLEMENTATION DEFINED mechanism to save AHA context.

For the OS overprovisioning example no remapping is required, and no trapping of management messages either since the same software is in charge of configuration and overprovisioning:

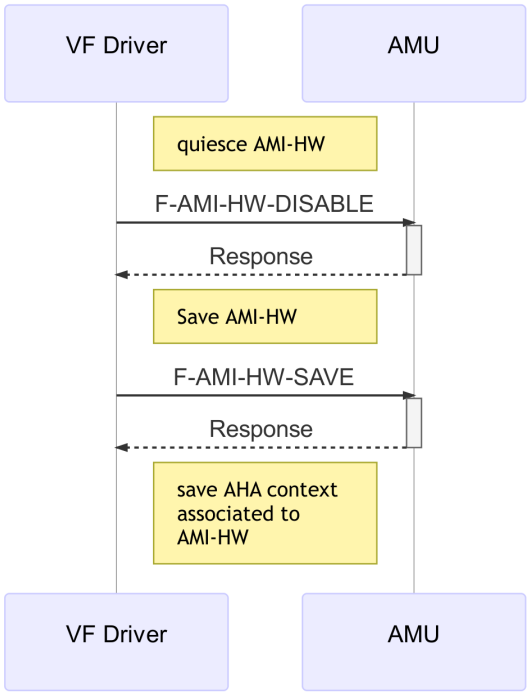

Figure C3.4: VF driver-AMU communication sequence diagram to prepare reassignment of AMI-HW.

C3.5.2 Disabling and saving the remapped AMI-SW

The following diagram shows command/response messages exchanged through the management AMI when disabling the remapped AMI-SW for hypervisor overprovisioning.

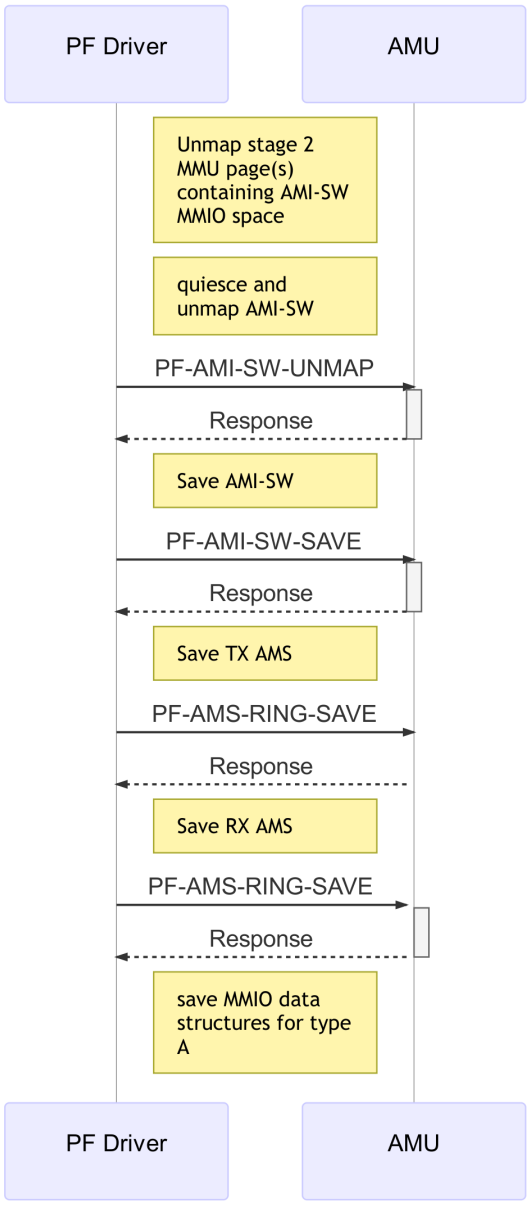

Figure C3.5: PF driver-AMU communication sequence diagram to prepare relocation of AMI-SW.

All AMI-SW accesses (data and commands) must be trapped for hypervisor to be notified of any VM activity dealing with overprovisioned AMI. Quiescing the AMI-SW will ensure there are no in flight messages associated with the AMI-SW when remapping it. PF driver also needs to save MMIO data structures in case of type A (Digest masks and AMS indices).

For the OS overprovisioning example no un-mapping is required:

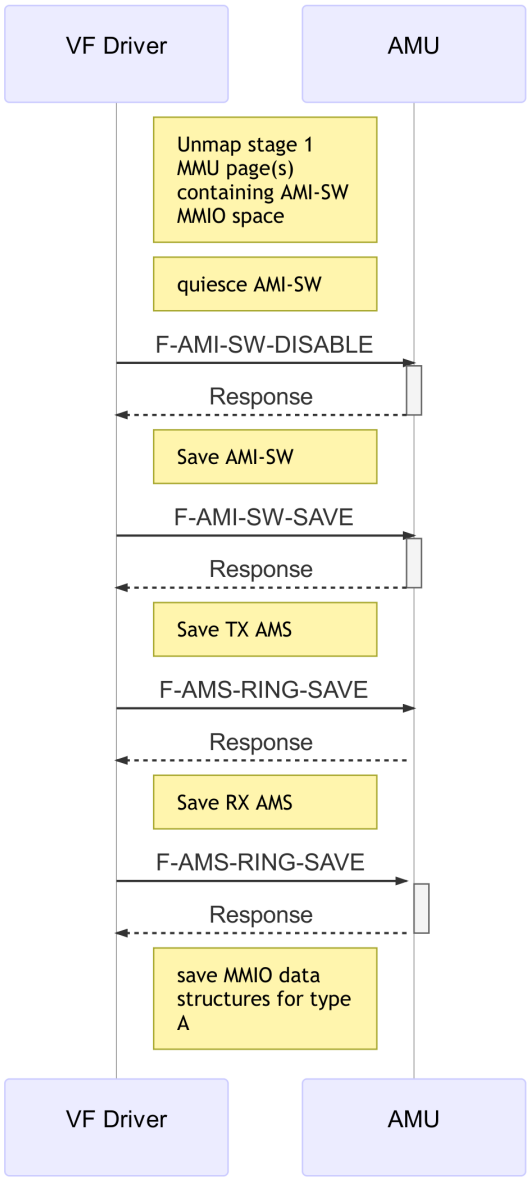

Figure C3.6: VF driver-AMU communication sequence diagram to prepare relocation of AMI-SW.

C3.5.3 Remapping/reassigning the AMIs

The following diagram shows command/response messages exchanged through the management AMI when relocating AMIs between VFs in different VMs. Note that to avoid generating spurious interrupts when restoring the indices we always map first the AMI in a DISABLED state in the PF to do so.

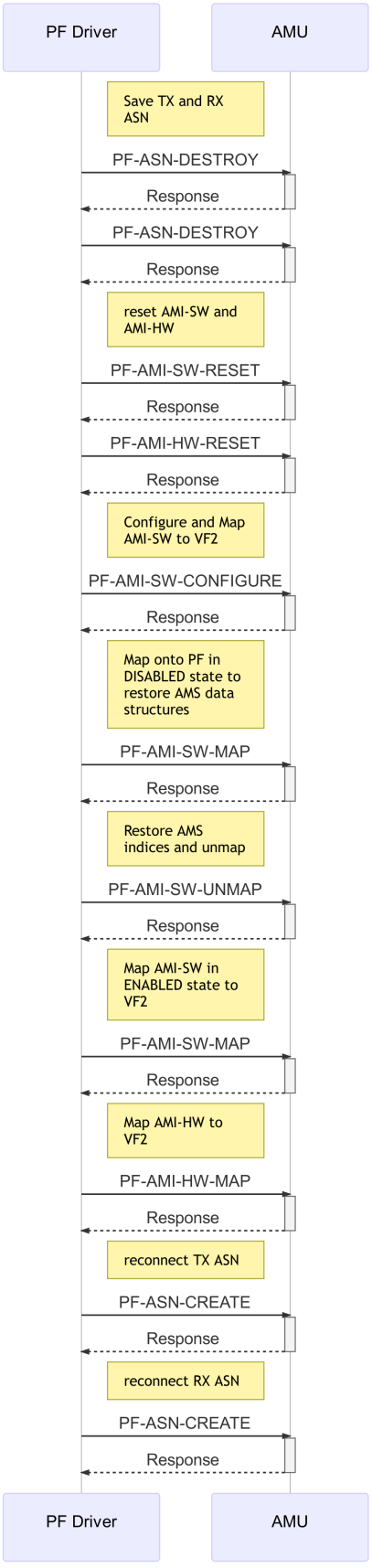

Figure C3.7: PF driver-AMU communication sequence diagram to relocate AMIs.

Once the AMIs are assigned to VF2 the system software can configure the AMI-HW and the AMI-SW according to VM2 application's needs, refer to Chapter C1 *Null accelerator* for an example.

Alternatively, if the VF2 AMIs were previously remapped due to overprovisioning policy, PF driver can restore the previous state using the proper command sequence before mapping the AMIs, in a similar way than in the live migration scenario. For further details refer to Chapter C2 *VM Live Migration*

In the case of OS overprovisioning there is no hypervisor intervention required unless ASN properties need to be changed:

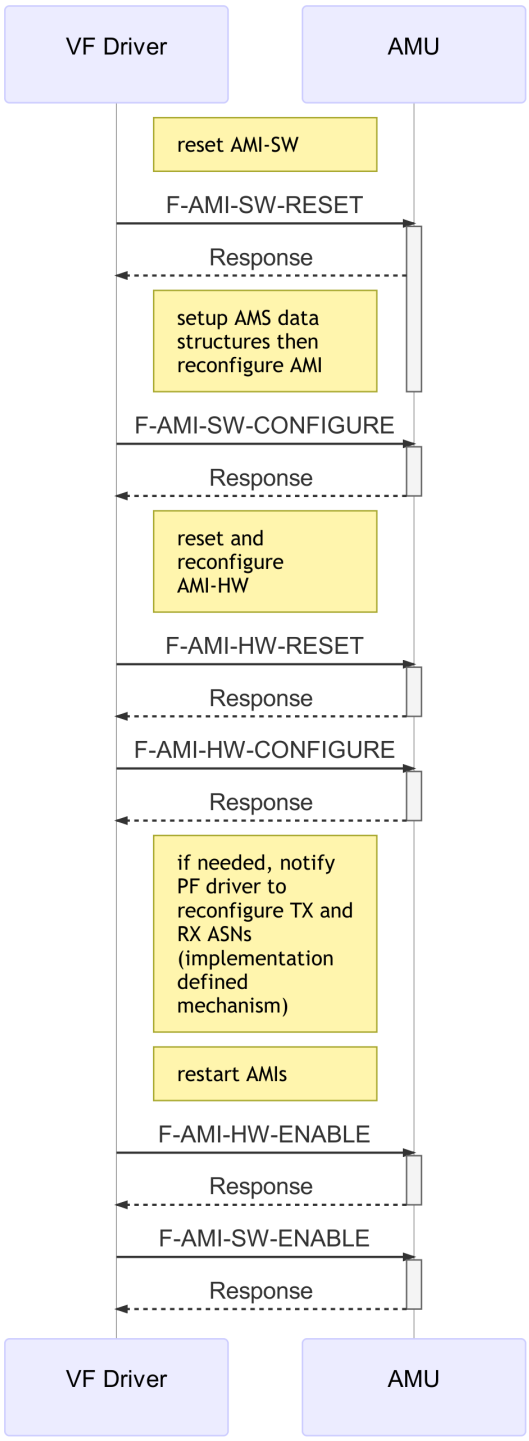

Figure C3.8: VF driver-AMU communication sequence diagram to relocate AMIs.

Chapter C4
AHA chaining

C4.1 Introduction

This example shows how to chain two accelerator contexts in a Revere-AMU system with a simple AHA using Message Format 0 (MFO0) messages with in-band data.

The device comprises a single AMU with two instances of the Null AHA (NAHA1 and NAHA2). This **NAHA (Null AHA)** has the following characteristics:

- It supports one hardware context
- The hardware context has an AMI-HW with a single RX AMS (for requests) and a single TX AMS (for responses)
- It has no DMA resource and each context has no associated state

The function of the NAHA is to simply return any request message on the RX AMS as a response on the TX AMS.

Two applications are running, one in VM1 sending messages and one in VM2 receiving them. For that purpose VF1 has been assigned to VM1 and VF2 has been assigned to VM2. Each VF has a single AMI-SW mapped onto it.

Each software driver has an AMI-SW with one AMS (request for the one mapped in VF1 and response for the one mapped in VF2). In this example each application is running in a different VM but they could run in the same one with no major modification to the scenario. Following picture displays the logical IDs (once mapped into the VFs).

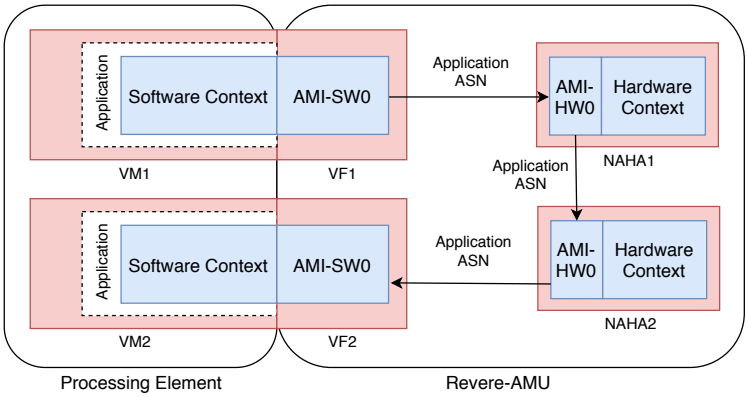

Figure C4.1: Logical view of the system showing the AMIs and ASNs

C4.2 Software architecture

In this example there are two software drivers:

One instance of a PF driver that configures the AMU including statically setting up the required ASNs.

Two instances of a combined VF and user driver that do both VF specific configuration and use the device by sending requests and receiving responses. These drivers are contained in separate virtual machines but they could be in the same with no major change.

There is no AHA specific configuration for the NAHA.

The device PF driver uses the management interface to manage the overall configuration of the system (including ASN creation and mapping of AMIs to Functions).

The VF driver 1 and the VF driver 2 use the management interfaces of their VFs to manage the configuration for VF1 and VF2 respectively. Device VF driver 1 and VF driver 2 also make use of physical AMI-SW1 and AMI-SW2 respectively to communicate with the NAHA1 and NAHA2 contexts (Request/Response).

Note

As detailed below in Table C4.1 and Table C4.2, all AMI-SWs and AMI-HWs are mapped with local ID zero in the two VFs.

C4.3 PF driver operation

C4.3.1 Binding the PF driver to the PF

Refer to C1.3 *PF driver operation*

C4.3.2 Virtual Function and AMI allocation

Refer to C1.3 *PF driver operation*

The following sequence diagram shows the messages that the PF driver exchanges with the AMU in order to perform this configuration:

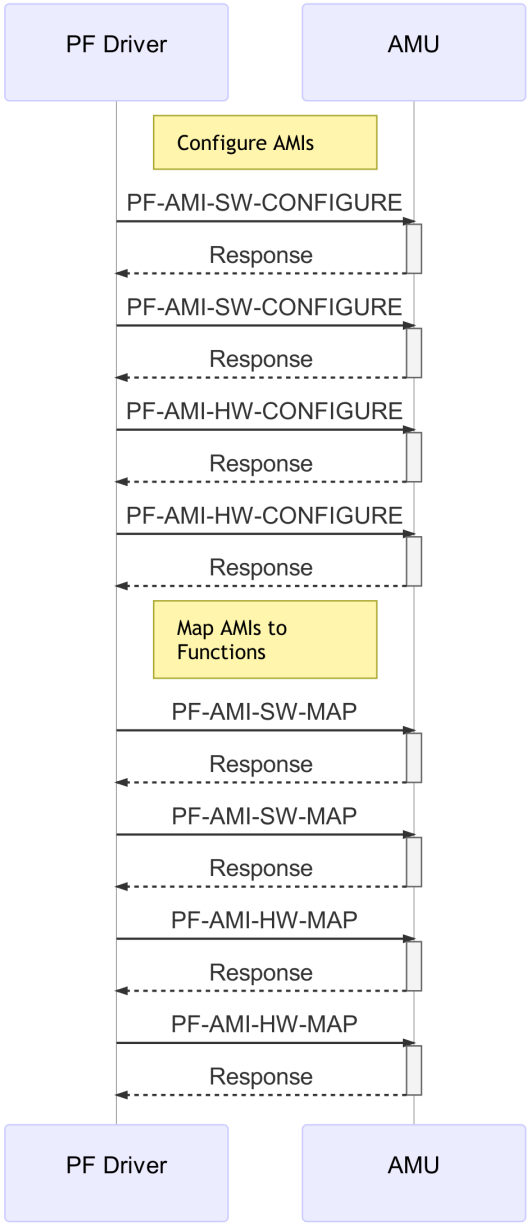

Figure C4.2: PF-VF Driver Communication Sequence diagram.

The following table shows the parameters used to map and configure the two AMI-SWs:

Table C4.1: AMI-SW mapping

Physical AMI-SW	Function Owner	AMI-SW ID	PASID
1	1	0	n/a
2	2	0	n/a

The following table shows the parameters used to map and configure the two AMI-HWs:

Table C4.2: AMI-HW mapping

Physical AMI-HW ID	AHA ID	Function Owner	AMI-HW ID	PASID	LOG2_HW_CRED_SIZE
1	0	1	0	n/a	1
2	1	2	0	n/a	1

C4.3.3 ASN allocation

Note that, at this point, no ASN has been established. The following ASNs are required per AMI-SW for this example:

- One ASN between the TX AMS in VF1's AMI-SW and RX AMS in NAHA1's AMI-HW.
- One ASN between the TX AMS in NAHA1's AMI-HW and RX AMS in NAHA2's AMI-HW.
- One ASN between the TX AMS in NAHA2's AMI-HW and RX AMS in VF2's AMI-SW.

Note

In this example, the role and number of ASN per AMI is known in advance, therefore it is possible for the PF driver to configure all needed ASNs at initialisation time. For other examples, the VF driver sends IMPLEMENTATION DEFINED messages to the PF driver via the AMU to request a specific configuration.

The PF driver requests the creation of ASNs through the management interface. The following sequence diagram shows the messages that the PF driver exchanges with the AMU in order to perform this configuration:

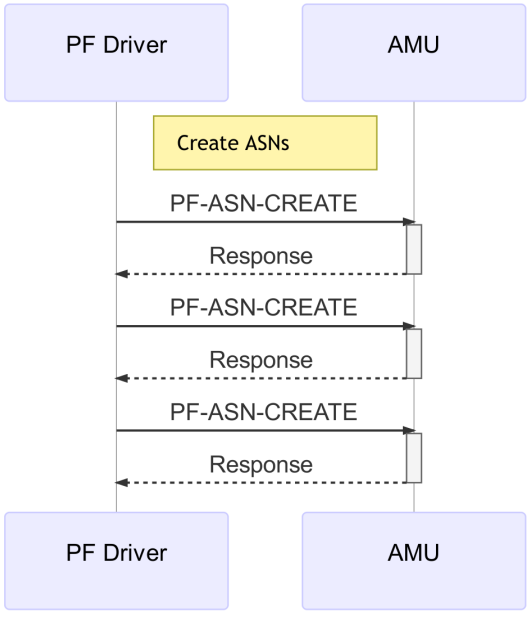

Figure C4.3: PF ASN creation Sequence diagram.

The following table shows the parameters used to create the three ASNs:

ASN ID	Message Format	Src. Owner	Src. AMI IDX	Src. AMS	Dst. Owner	Dst. AMI IDX	Dst. AMS	MPIDR
0	0	1	AMI-SW0	0	1	AMI-HW0	0	-
1	0	1	AMI-HW0	0	2	AMI-HW0	0	-
2	0	2	AMI-HW0	0	2	AMI-SW0	0	-

C4.4 VF driver operation

C4.4.1 Binding the VF and user driver to the VF

A VF driver running on the host machine is bound to the Virtual Function of the device (AMU and associated AHA).

System software binds the combined VF and user driver by matching the Device ID of the Virtual Function. Once the VF driver is loaded, it reads the capabilities registers and performs initialisation.

C4.4.2 AMS Allocation

The PF driver has appropriately configured AMIs and ASNs required for this example. The last step to get the ASN to a functional state is to configure the AMSes appropriately. This is the responsibility of the VF driver.

The following diagram shows the messages that each VF driver exchanges with the AMU to configure the AMSes.

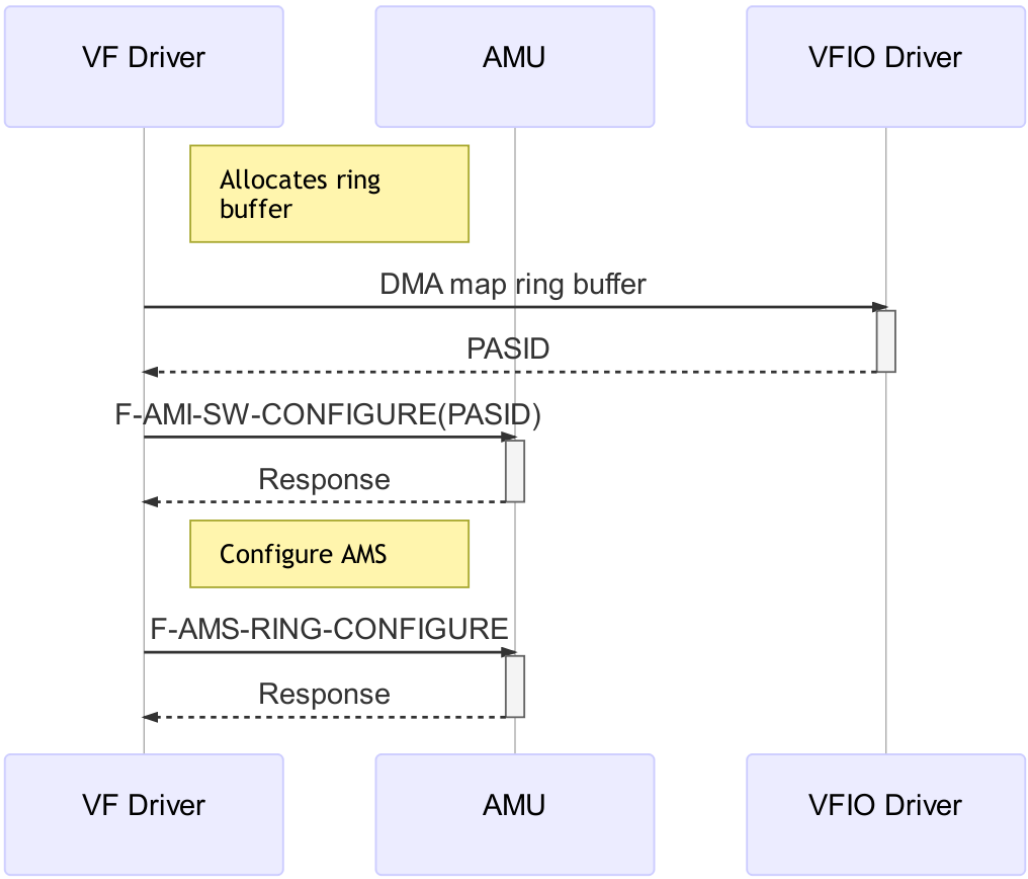

Figure C4.4: VF Driver AMS Configuration Sequence diagram.

The following table shows example parameters used to configure the AMSs:

Table C4.4: AMI-SW configuration parameters

LOG2_SLOT_SIZE	LOG2_SIZE	RX_MODE	THRESHOLD	RING_BASE_PTR
3	3	BACK_PRESSURE	0	-

The RING_BASE_PTR must be set according to the translation scheme that system software configures for the Virtual Function.

In this example, the applications sending and receiving the messages would allocate memory the for the AMS ring buffer.

VF driver will allocate a buffer in guest virtual memory (address VA), and use VFIO driver ioctl to have the system software map the buffer in the SMMU.

The index must be initialised to 0. The remaining parameters are not critical for the purposes of this example and can be modified as needed.

Each of the Virtual Function drivers can now access the AMI-SW interfaces assigned to them through the VF BAR space.

At this stage the system has been fully configured and the VF1 driver can send messages.

C4.5 Application

Any message pushed to TX AMS 0 in VM1 will reach the NAHA1 through the associated AMI-HW, which will transfer the message to RX AMS 0 in NAHA2. NAHA2 will then forward it to RX AMS0 in VM2.

It is IMPLEMENTATION DEFINED how the communication between the NAHAs is implemented and how the routing of messages is performed.

A more configurable AHA could also support several AMI-HWs along with configurable routing between AMSes. In that case it is IMPLEMENTATION DEFINED how this routing internal to the AHA is configured.

Chapter C5
Lightweight

Will be added in a future release.

Chapter C6
Virtualization offload

Will be added in a future release.

Chapter C7
Networking SoC

Will be added in a future release.

Chapter C8
Wake-on-LAN

C8.1 Introduction

This example shows how a network interface AHA can be used to wake up the system from low power mode in response to an incoming network packet with a specific payload (a.k.a. “magic packet”).

In this example the system is first moved to a low power state. A “magic packet” is then received by the network interface AHA, which triggers a wakeup event. Finally, the system is resumed.

Figure C8.1 gives an overview of this use case:

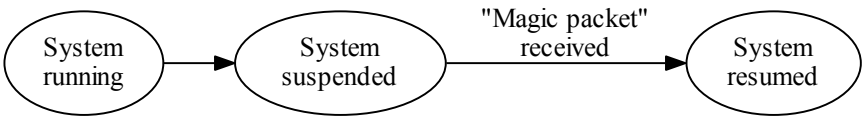

Figure C8.1: Wake-on-LAN use case

The device comprises a Revere-AMU with a network interface AHA.

The Revere-AMU has the following characteristics:

- It is an on-chip Type A2 AMU

- All its Functions implement the Power Management Capability and support the D0 and $D3_{hot}$ power states, and all its Functions have their No_Soft_Reset bit clear, do not retain their state in $D3_{hot}$ and therefore necessitate initialization after transitioning out of $D3_{hot}$
- It has IMPLEMENTATION DEFINED On and Off power states, which support the Functions D0 and $D3_{hot}$ power states (see Table C8.3)
- Its Physical Function supports generating PME in $D3_{hot}$ power state

The network interface AHA has the following characteristics:

- It supports a single hardware context
- Its hardware context has a single AMI-HW with one TX AMS and one RX AMS to exchange network packets
- It has no DMA resources
- It has IMPLEMENTATION DEFINED On and Off power states, which support the Functions D0 and $D3_{hot}$ power states (see Table C8.4)
- It can receive network packets, detect a specific payload ("magic packet") and signal an always-on power domain wake event to the System Power Controller when in Off power state

The network interface AHA relays any packet it receives from the network as a Revere message. It also relays any Revere message it receives as a packet to the network.

Figure C8.2 shows the logical view of the system, with configured ASNs between the HW accelerator context and a software driver:

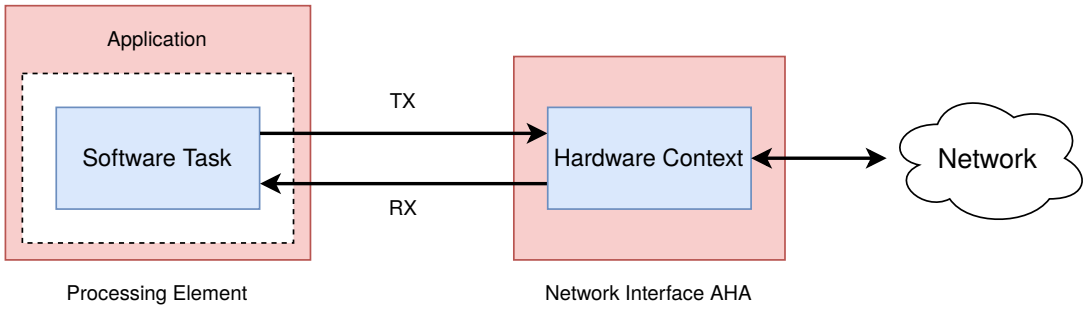

Figure C8.2: Logical Wake-on-LAN system view

The device of this example is integrated in a system. This system has the following characteristics:

- It has a single PE, which supports low power modes
- It has a Power Controller, which can wakeup the system from low power mode and can duplicate wakeup events as IMPLEMENTATION DEFINED interrupts to the GIC
- It has a GIC, which supports low power modes

Figure C8.3 shows a more power centric view of the system, with the power domains described as in SBSA [3]:

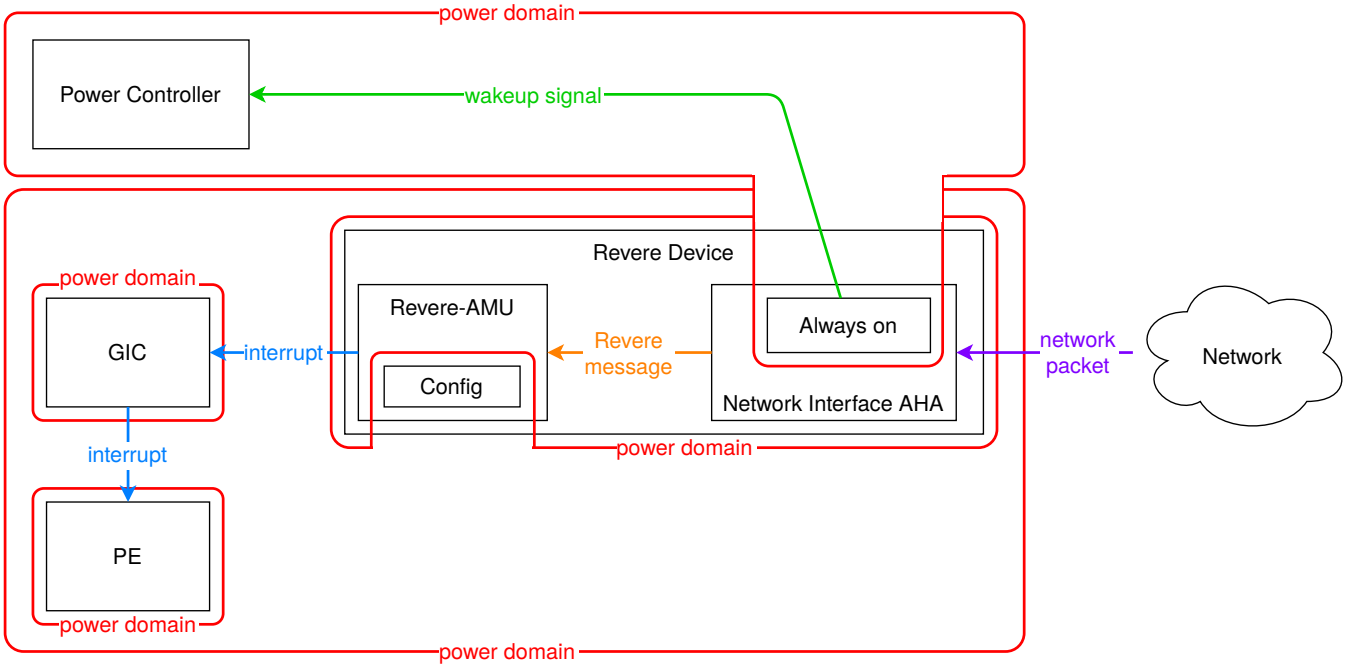

Figure C8.3: Wake-on-LAN system block diagram

C8.1.1 Packet and message formats

In this example, the network interface AHA uses Message Format 1 (MFO1) to convey network packets as payload of Revere messages. The network packets of this example are ethernet frames with user data of variable size, with maximum size 1500B.

Figure C8.4 shows the format of the Revere messages exchanged with the network interface AHA:

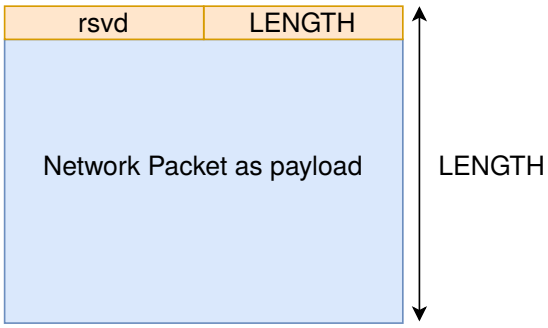

Figure C8.4: Message Format

Table C8.1 details the Revere message format used by the network interface AHA of this example to transport network packets, ethernet frames:

Table C8.1: Network interface AHA Revere message format

Bytes	Description
8	MFO1 Descriptor, comprising LENGTH
6	Destination MAC address

Bytes	Description
6	Source MAC address
2	Type / Length
46-1500	User Data
4	Frame Check Sequence

Table C8.2 details the format of the user data for the "magic packet" used in this example:

Table C8.2: "Magic packet" payload format

Bytes	Description
6	Synchronization bytes with value 0xFF
96	Destination MAC address repeated 16 times

C8.1.2 Power states

We detail the Revere-AMU and network interface AHA power states for this example. Those IMPLEMENTATION DEFINED power states are similar to the power states in SBSA [3]. They support the implemented PCI Power States.

Table C8.3 lists the Revere-AMU power states for this example:

Table C8.3: Revere-AMU power states

AMU power state	Supported PCI Power State	Description
On	D0	The Revere-AMU is powered up.
Off	$D3_{hot}$	The Revere-AMU is powered down. It cannot transport Revere messages. No state is retained except the PME context. Accesses to the Configuration Space from the PE are still possible. Transition to the On state must be initiated by writing to the PMCSR.

Table C8.4 lists the network interface AHA IMPLEMENTATION DEFINED power states for this example:

Table C8.4: Network interface AHA power states

AHA power state	Supported PCI Power State	Description
On	D0	The network interface AHA is powered up.
Off	$D3_{hot}$	The network interface AHA is powered down, except the logic related to "magic packet" detection. It cannot transport Revere messages. No state is retained except state related to PME context such as wake event configuration.

AHA power state	Supported PCI Power State	Description
		Transition to the On state follows Revere-AMU power state transition. A wakeup event can be signalled to the Power Controller.

In this example, the Revere device as a whole has a single power state and the Revere-AMU and network interface AHA power states are linked together. The Revere-AMU and the network interface AHA can either be both in their On power state or both in their Off power state.

C8.2 Initial setup

In this example there are two software drivers:

One instance of a PF driver that configures the AMU including statically setting up the required ASNs. The AHA also requires IMPLEMENTATION DEFINED per-context configuration. This configuration is also performed by the PF driver.

One instance of a combined VF and user driver that do both VF specific configuration and exchanges messages with the network interface AHA. This driver is contained within a Virtual Machine.

Figure C8.5 shows a view of the complete system detailing the mapping of software drivers and AMI-SWs to PCI Functions and ASNs configured between the AMI-SW0 and the network interface AHA:

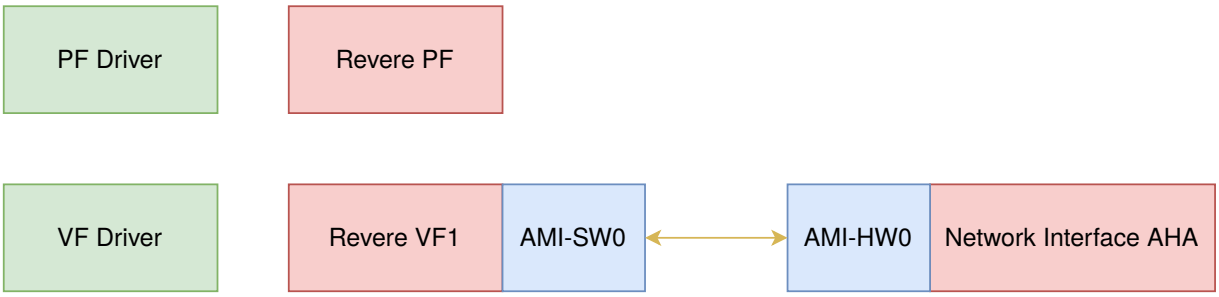

Figure C8.5: Software Mapping

The device PF driver uses the management interface to manage the overall configuration of the system (including ASN creation and assignment of AMI-SW to Functions).

The device VF driver uses the management interface to manage the configuration for VF1. Device VF driver also makes use of AMI-SW0 to communicate with a context in the network interface AHA.

C8.3 Starting point and prerequisites

This example assumes that:

- A single VF has been created.
- AMIs have been mapped.
- ASNs between the AMIs have been created.
- Both the PF and VF are bound to a PF driver running on the host and a VF driver running inside a VM respectively.

For the details on how this initial setup is performed, refer to Chapter C1 *Null accelerator*.

Additionally:

- The SMMU has been configured to translate accesses from VF1 to the address space of the virtual machine.
- An IMPLEMENTATION DEFINED mechanism has been configured by the PF driver to offer a suspend service to the VM.
- An IMPLEMENTATION DEFINED mechanism has been configured by the platform firmware to offer a suspend service to the PF driver.
- The PF and VF are in $D0_{active}$ PCI Power State and the AMU and network interface AHA are in their On power states.
- The stage 2 MMU page table of the PE has been setup so that accesses to PCI Config Space from within the VM are trapped by the PF driver.
- The PF PMCSR PME_En field has been set to 1 to enable the generation of PME, and in this example to enable generation of always-on power domain wake events by the network interface AHA.

Note

An example mechanism the PF driver can use to offer services to the VM is to handle `HVC` instructions issued by software from within the VM. The SMC Calling Convention (SMCCC) [9] is an example of calling convention that can be used for that purpose.

An example of VM suspend service that can be provided by the PF driver is the `SYSTEM_SUSPEND` of the Power State Coordination Interface (PSCI) [10]. It can be called using an `HVC` "conduit" conforming to the SMCCC.

The same kind of service and mechanism can be provided by the platform firmware to the PF driver: a PSCI [10] `SYSTEM_SUSPEND` service using an `SMC` conduit, conforming to the SMCCC [9].

C8.3.1 Detailed configuration

This section lists the detailed Revere-AMU configuration for this example. For the details on how this initial setup is performed, refer to Chapter C1 *Null accelerator*.

Table C8.5 shows the parameters used to map and configure the AMI-SW0:

Table C8.5: AMI-SW parameters

Parameter	Value
Physical AMI-SW	0
Function Owner	1
AMI-SW ID	0
PASID Enable	0

Parameter	Value
PASID	-
Rx IRQ Enable	0
Tx IRQ Enable	0
Rx Vector	-
Tx Vector	-
Enable	1

Note

Type B parameters are omitted from the table.

See PF-AMI-SW-MAP and PF-AMI-SW-CONFIGURE for details.

In this example, the AMI-SW0 rings are configured to contain only one slot for simplicity. Table C8.6 shows the parameters used to configure the two AMSes of AMI-SW0:

Table C8.6: AMSes parameters

Parameter	RX AMS	TX AMS
Function Owner	1	1
AMI-SW ID	0	0
AMS Type	0	1
AMS ID	0	0
Threshold	0	0
Ring Log_2 Size	0	0
Rx Mode	0	-

Note

The Base Pointer parameters for the rings are omitted from the table but must be configured to valid addresses within the virtual machine address space.

See PF-AMS-RING-CONFIGURE for details.

Table C8.7 shows the parameters used to map and configure the AMI-HW0:

Table C8.7: AMI-HW parameters

Parameter	Value
AHA ID	0
Physical AMI-HW	0
Function Owner	1
AMI-HW ID	0
Hardware Credit Size	1
PASID Enable	0
PASID	-
Enable	1

See PF-AMI-HW-MAP and PF-AMI-HW-CONFIGURE for details.

Table C8.8 shows the parameters used to create the two ASNs, with ASN0 for TX to network and ASN1 for RX from network:

Table C8.8: ASNs parameters

Parameter	First ASN	Second ASN
Source AMI Type	0	1
Destination AMI Type	1	0
Source Function Owner	1	1
Source AMI	0	0
Destination Function Owner	1	1
Destination AMI	0	0
Source AMS	0	0
Destination AMS	0	0
ASN ID	0	1
Log_2 Message Length	8	8
Message Format	1	1

Note

QoS, Monitoring, Tracing, Stashing, Profiling and Out-of-band Buffers parameters are omitted from the table for simplicity.

See PF-ASN-CREATE for details.

C8.4 Suspend operations

We describe the sequence of operations of the Software Task and PF driver, to move the system to low power mode.

In this example, the Software Task running within the VM moves its VF to $D3_{hot}$ by writing to the configuration registers. The PF driver traps and emulates those registers accesses and assists by quiescing the VF.

The Software Task then requests VM suspend. The PF driver reacts to VM suspend by moving the PF to $D3_{hot}$ and suspending the complete system.

Figure C8.6 gives an overview of the step involved in suspending the system:

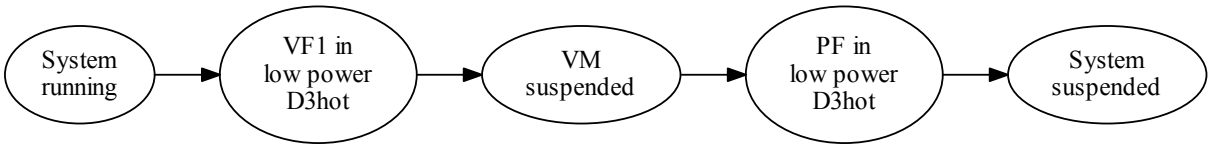

Figure C8.6: Suspend steps

We list the steps leading to system suspend in more details:

1. Software Task switches VF1 to $D3_{hot}$
   a. Software Task writes to VF1 PMCSR PowerState
   b. Access to the configuration space is trapped by the PF driver as a stage 2 MMU fault
   c. The PF driver quiesces VF1
      i. The PF driver disables VF1 with a PF-F-DISABLE command
      ii. The PF driver writes to VF1 BME, polls Transaction Pending
   d. Access to the VF1 PMCSR is relayed by the PF driver to the hardware
2. Software Task initiates VM suspend
   a. Software Task calls the IMPLEMENTATION DEFINED PF driver service to suspend the VM activity
   b. The PF driver suspends the VM
3. The PF driver opportunistically initiates system suspend
   a. The PF driver switches the PF to $D3_{hot}$
      i. The PF driver quiesces the PF by writing to BME, polls Transaction Pending
      ii. The PF driver writes to PF PMCSR PowerState, preserving PME_En. The PF loses its functional context and VF1 no longer exists. The Revere-AMU and the network interface AHA enter their Off power states.
   b. The PF driver requests system suspend to the platform firmware using an IMPLEMENTATION DEFINED mechanism
   c. Platform firmware suspends the system

Note

Example of IMPLEMENTATION DEFINED suspend services are detailed in C8.3 *Starting point and prerequisites*.

Synchronization barriers after register writes are omitted for simplicity.

For more details on quiescing, refer to Chapter C2 *VM Live Migration* and Chapter C3 *Overprovisioning*.

Table C8.9 lists the power states for each step of the suspend operations:

Table C8.9: Power states when suspending the system

Step	PE	VM	PF	VF1	AMU	AHA	Description
1	Run	Running	$D0_{active}$	$D0_{active}$	On	On	
1d	Run	Running	$D0_{active}$	$D3_{hot}$	On	On	VF1 enters low power $D3_{hot}$
2b	Run	Suspended	$D0_{active}$	$D3_{hot}$	On	On	VM enters low power suspend
3aii	Run	Suspended	$D3_{hot}$	-	Off	Off	AMU and AHA enter low power Off
3c	Off	Suspended	$D3_{hot}$	-	Off	Off	System enters low power suspend

Refer to SBSA [3] for PE power states details.

Note

When VF1 enters low power, AMI-SW0 and AMI-HW0 inherit the $D3_{hot}$ Power State as they are mapped into VF1. In this example this has no effect on the hardware for simplicity. Advanced Revere-AMU implementations and associated AHAs could seize the opportunity and reduce some power consumption at this step already.

The following sequence diagrams describe the steps taking the system to low power mode:

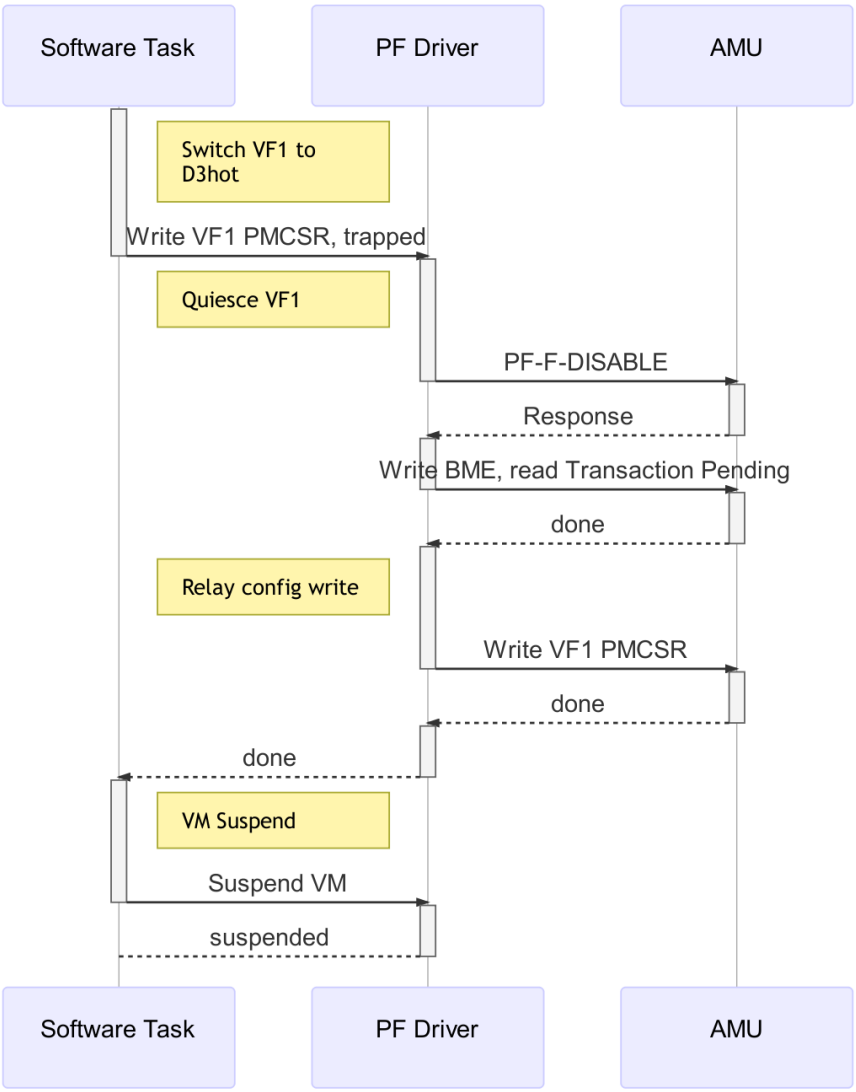

Figure C8.7: VM suspend sequence diagram

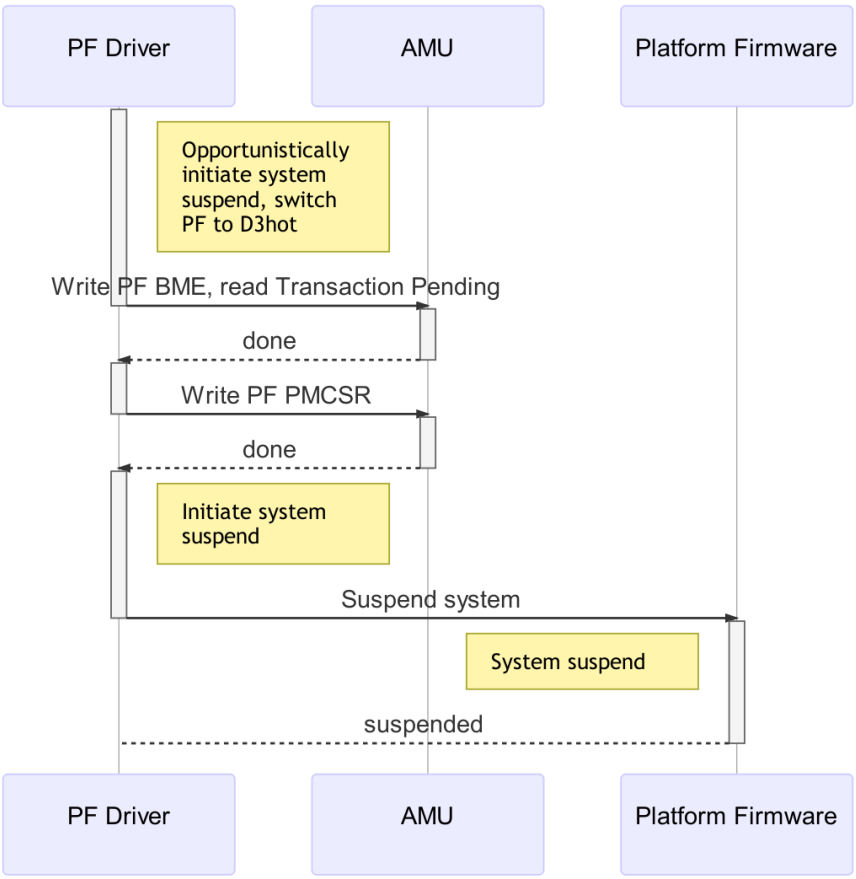

Figure C8.8: System suspend sequence diagram

At the end of the suspend operations the system is in low power state and the network interface AHA is ready to detect an incoming “magic packet” from the network.

C8.5 Wake-up events

We describe the sequence of hardware events triggering a system wake-up from low power mode.

Figure C8.9 gives an overview of the chain of hardware events involved:

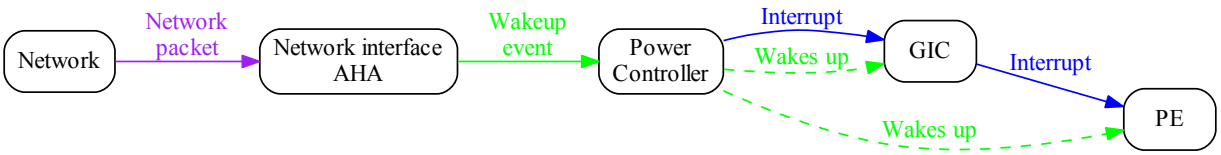

Figure C8.9: Hardware events chain

The chain of hardware events is the following:

1. A “magic packet” is received by the network interface AHA
2. The network interface AHA signals a wakeup event to the Power Controller
3. The Power Controller duplicates the wakeup event as an interrupt to the GIC
   a. The Power Controller wakes up the GIC
   b. The Power Controller wakes up the PE
4. The PE comes out of reset and software restores GIC state, handles the interrupt
   a. Software can proceed resuming the system (see C8.6 *Resume operations*)

Table C8.10 lists the power states for each step of the chain of hardware events:

Table C8.10: Power states for waking up with a wakeup event

Step	PE	GIC	AMU	AHA	Description
1	Off	Off	Off	Off	
3a	Off	On	Off	Off	GIC back to On state
3b	Run	On	Off	Off	PE comes out of reset

Refer to SBSA [3] for PE and GIC power states details.

At the end of the wake-up operations the system has been moved out of low power mode by the reception of a “magic packet” and software is proceeding with system resume.

Note

The network “magic packet” is lost.

C8.6 Resume operations

We describe the sequence of operations of the Software Task and PF driver, to resume the system from low power mode and move back to running mode. Those operations correspond to the inverse sequence of the one described in C8.4 *Suspend operations*.

In this example, the PF driver is resumed after system wakeup. It resets the PF and moves it to $D0_{uninitialized}$ then $D0_{active}$, performs device reconfiguration, moves the VF1 to $D3_{hot}$ and finally resumes the VM containing the Software Task.

The Software Task running within the VM resets its VF and moves it to $D0_{uninitialized}$ then $D0_{active}$ by writing to the configuration registers. The PF driver traps and emulates those registers accesses.

Finally, the Software Task performs VF1 reconfiguration and resumes operations.

Note

The network “magic packet” is lost.

Figure C8.10 gives an overview of the step involved in resuming the system:

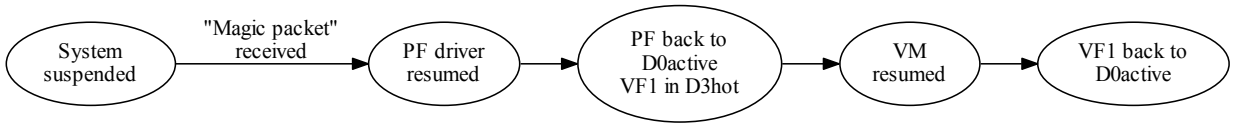

Figure C8.10: Resume steps

We list the steps of system resume in more details:

1. PF driver handles System Controller interrupt
   a. The PF driver requests full system resume to platform firmware using the IMPLEMENTATION DEFINED mechanism
   b. Platform firmware resumes full system
2. PF driver takes the device out of low power Off
   a. PF driver performs PF FLR
   b. PF driver switches PF to $D0_{active}$ and reconfigures it
   c. PF driver reconfigures the VFs, VF1 is in $D0_{uninitialized}$
   d. PF driver switches VF1 to $D3_{hot}$ by writing to VF1 PMCSR PowerState
3. PF driver performs device reconfiguration
   a. AMI-HW0 is reconfigured and mapped with PF-AMI-HW-CONFIGURE and PF-AMI-HW-MAP commands
   b. AMI-SW0 is mapped with a PF-AMI-SW-MAP command
   c. Sessions are created with two PF-ASN-CREATE commands
4. PF driver resumes VM activity
5. Software Task takes VF1 out of low power $D3_{hot}$
   a. Software Task performs VF1 FLR
   b. Software Task switches VF1 to $D0_{active}$ and reconfigures it. Accesses to the configuration space are trapped by the PF driver as a stage 2 MMU fault. Original accesses to the VF1 configuration registers are relayed by the PF driver to the hardware.
6. Software Task performs VF1 reconfiguration

a. AMI-HW0 is reconfigured with F-AMI-HW-CONFIGURE and F-AMI-HW-ENABLE commands
b. AMI-SW0 is reconfigured with F-AMI-SW-CONFIGURE, F-AMS-RING-CONFIGURE and F-AMI-SW-ENABLE commands

Note

Revere-AMU Functions might not be ready immediately when resuming to D0, in which case the software needs to wait before it can access the Function.

In this use case, the Revere-AMU Functions retain only their context related to PME when in $D3_{hot}$ power state. Software has to perform Function reconfiguration in that case.

Synchronization barriers after register writes are omitted for simplicity.

If more network packets are received during system resume, they are lost.

For more details on how reconfiguration is performed, refer to Chapter C1 *Null accelerator*.

The PF driver also has to perform AMI-HW configuration even though the Software Task will do it, as only the PF-AMI-HW-CONFIGURE command can modify the LOG2_HW_CRED_SIZE parameter.

In this scenario the Software Task expects VF1 in $D3_{hot}$ after VM resume. Other conventions between the PF driver and the Software Task are possible.

Table C8.11 lists the power states for each step of the resume operations:

Table C8.11: Power states when resuming the system

Step	PE	VM	PF	VF1	AMU	AHA	Description
1	Run	Suspended	$D3_{hot}$	-	Off	Off	
2a	Run	Suspended	$D0_{uninitialized}$	-	On	On	AMU and AHA back to On
2b	Run	Suspended	$D0_{active}$	-	On	On	PF reconfigured
2c	Run	Suspended	$D0_{active}$	$D0_{uninitialized}$	On	On	VFs reconfigured
2d	Run	Suspended	$D0_{active}$	$D3_{hot}$	On	On	VF1 enters low power $D3_{hot}$
4	Run	Running	$D0_{active}$	$D3_{hot}$	On	On	VM resumed
5a	Run	Running	$D0_{active}$	$D0_{uninitialized}$	On	On	VF1 reset
5b	Run	Running	$D0_{active}$	$D0_{active}$	On	On	VF1 reconfigured

Refer to SBSA [3] for PE power states details.

The following sequence diagrams describe the steps resuming the system out of low power mode:

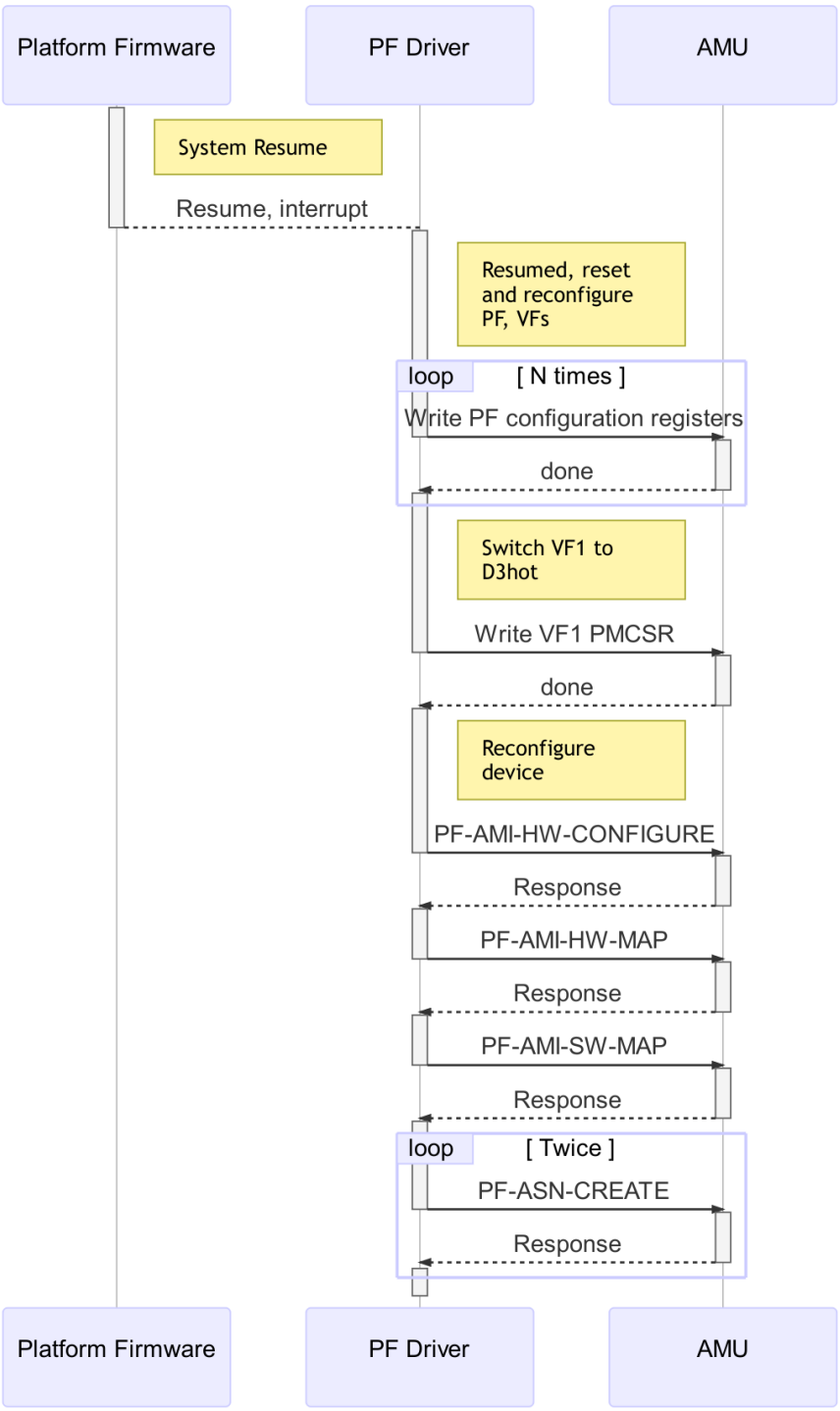

Figure C8.11: System resume sequence diagram

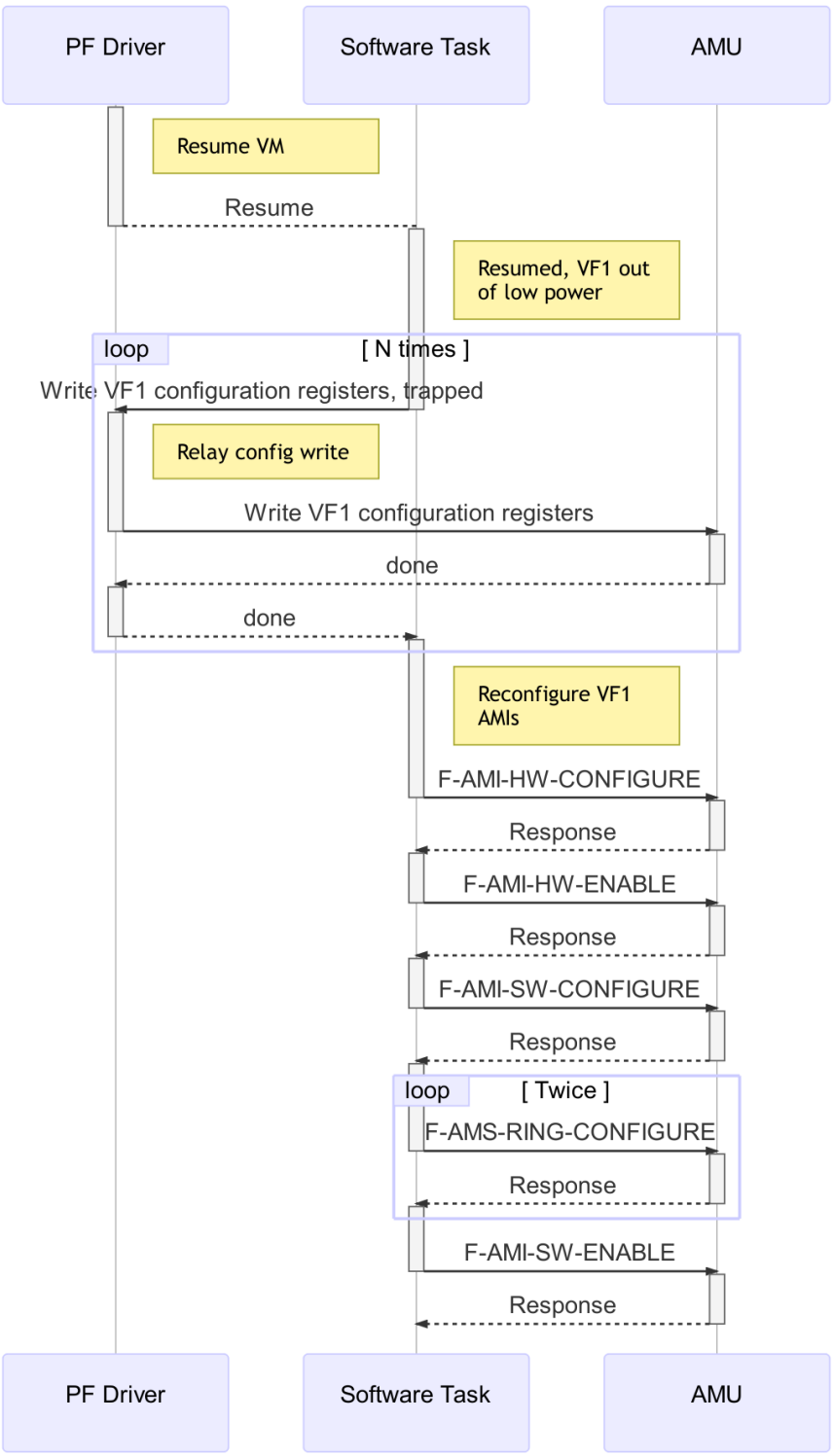

Figure C8.12: VM resume sequence diagram

At the end of the resume operations the system has been resumed to running state.

Glossary

AHA

Accelerator management unit Hardware Agent
A physical entity in the system that sends/receives messages. Examples: a programmable or fixed Function accelerator. AHA optionally have an IMPLEMENTATION DEFINED mechanism for managing multiple contexts and use a logical AMI-HW per context.

AMI

Accelerator management unit Message Interface
An AMI is a logical grouping of AMS used by a single context to send and receive messages.

AMI-HW

Accelerator management unit Message Interface for Hardware

AMI-SW

Accelerator management unit Message Interface for Software

AMS

Accelerator management unit Message Socket
Part of an AMI used for sending/receiving messages. An AMS is the endpoint of a Accelerator Session (either for transmit or receive).

AMS-RX

Receive Accelerator management unit Message Socket

AMS-TX

Transmit Accelerator management unit Message Socket

AMU

Accelerator Management Unit

ASN

Accelerator management unit Session
Connection between an AMS-TX and an AMS-RX for ordered, credited transport of messages. A session is an established unidirectional AMS to AMS flow of messages between exactly two AMSs. An Accelerator Session is independently credited end to end and is ordered. Crediting means that a message will leave an AMS-TX only if the AMS-RX has provisioned enough space to receive the message.

Buffer

A buffer is a set of contiguous data mapped in a tasks virtual memory space defined by a pointer to the start of the buffer and a length.

Cache Stashing

Ability of an agent in the system to request that a line is allocated in (or stashed) to a cache local to another agent in the system.

Context

A context is a view of memory combined with associated state. For software running on an Armv8 PE, the context is identified by a VMID, an ASID and the PE architectural state for a thread.

CSM

Coherent Shared Memory

Descriptor

A descriptor is a data structure that contains a command and data associated with a message. The data includes zero or more buffers each identified in the descriptor by a pointer and length.

Device

A device is a granule of hardware resources (for example: accelerators, I/O) in the system that is visible to software and can be managed independently. In the context of an AMU, a device is a combination of an AMU and all AHAs associated with that AMU. A device has memory mapped registers, may generate interrupts and may have DMA capabilities.

DMA

Direct Memory Access
Direct Memory Access performed by a device.

DMA Engine

A specific AHA that performs data transfers between a source buffer and a destination buffer.

Doubleword

A 64-bit data item. Doublewords are normally at least word-aligned in Arm systems.

Doubleword-aligned

Means that the address is divisible by 8.

FLR

Function Level Reset, as per the PCI Express Base Specification [2].

Message

A message is a logical entity comprising a command and associated data.

PME

Power Management Event, as per the PCI Express Base Specification [2].

QoS

Quality of service

Table

A table is an array data structure in normal memory that is shared between software and hardware, has a defined element structure, is identified by a starting address and size (number of entries) and has a defined mechanism for maintaining coherency between software and hardware.